SOFTWARE MANUAL

EKF Intelligent I/O Controller Family On CompactPCI

Document No. 2468
Edition 2 Released in July 2001

Contents

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Tables

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

About this Manual

This manual describes the technical aspects of the Windows NT4/2000 driver and the board level interface to members of EKF's Intelligent I/O Controller family, required for installation and system integration. It is intended for system administrators, and for driver and application writers only.

Edition History

EKF Document "ekf960mle.wpd" Text #2468			
Ed.	Changes	Author	Date
1	1st Edition of the Software Manual English reflecting version 1.30.0.0 of the WinNT/2000 driver and version 1.21 of the firmware.	gn	July 2001
2	Added description of new fields in structures SJA1000PERF_STATS and SJA1000_STATUS. Added new defines SJA1000_EV_BUSOFF, SJA1000_EV_BUSON and SJA1000_ERROR_BUSOFF. This reflects version 1.31.0.0 of the WinNT/2000 driver and version 1.22 of the firmware.	gn	2001-07-16

Trade Marks

Some terms used herein are property of their respective owners, e.g.

i960 RP: ® Intel
CompactPCI: ® PICMG
Windows 98, Windows NT, Windows 2000: ® Microsoft

EKF does not claim this list to be complete.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Legal Exclaimer - Liability Exclusion

This manual has been edited as carefully as possible. We apologize for any potential mistake. Information provided herein is designated exclusively to the proficient user (system integrator, engineer). EKF can accept no responsibility for any damage caused by the use of this manual.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Windows NT4/2000 Driver Description

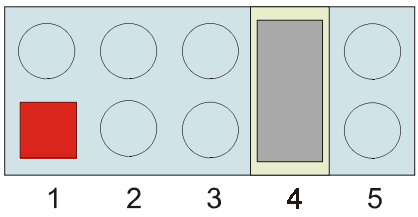
This chapter describes the requirements and features of the Windows NT4/2000 device driver “ekf960si1" for the EKF Intelligent I/O Controller family. Furthermore all issues related to the installation on a Windows NT4 respective Windows 2000 system are discussed.

Hardware Requirements

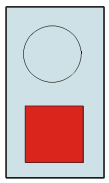
The installation of the WinNT/2000 driver “ekf960si1" requires a *CompactPCI* System that at least consists of

- main CPU Board with System Controller Function,
- at least one free *CompactPCI* slot for EKF's Intelligent I/O Controllers CG1-RADIO, CU1-CHORUS, CU2-QUARTET or CX1-BAND.

In order to get a proper function of the I/O controllers, they should have the following jumper configuration:



Configuration Jumper Field
JCNF



Clock
Source
Selector
JCLK

Note: The 5st jumper of **JCNF** exists on the CANbus controller board CX1-BAND only and replaces the function of jumper **JCLK**.

See also the *User's Manual* of the corresponding boards for details of the jumper settings and their function. It also contains a section how to install or remove a board into a *CompactPCI* system rack.

Some system BIOS's may have a problem to recognize the I/O controllers on system boot. When the controller isn't shown in the PCI device list that the BIOS displays during the boot procedure, pull jumper 4 of JCNF and reboot again. This will inhibit a board reset when the *CompactPCI* reset line is active. Disadvantage of this setting is, that power has to switch off and on again to generate an on-board reset on the I/O controller.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

If the board isn't still found after removing this jumper, check whether your system CPU board supports the so-called spread spectrum clocks on the *CompactPCI* bus. The clock in this case is modulated by another low frequent clock (typical 0.5% of the base clock) with the advantage of an improved EMC behaviour.

Since the EKF Intelligent I/O Controller Family boards are equipped with a PLL based clock buffer, the boards doesn't work properly when fed with a spread spectrum clock. Thus it is necessary to disable the clock spreading. Perhaps there exists a switch in the BIOS CMOS setup to do so. Ask your system CPU manufacturer for details if there is any unclarity about this issue.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Software Requirements

Board Firmware

To use the ports on the I/O controllers, the local firmware on the adapters must run. See section “**Hardware Requirements**” above for correct jumper setting to make sure that. The driver “ekf960si1" checks whether the firmware responses to messages sent to it and whether the version of this firmware is proper to work with it. The driver won't start and creates a log entry (viewable with the EventViewer) if one of these conditions fail.

Boards of EKF’s Intelligent I/O Controllers are delivered with the last recent version of the firmware. Nevertheless a copy of the actual version of the I/O controller firmware is included within each installation pack as well as a programming tool to permanently load it down to the I/O controller.

Firmware Update

The driver setup tool provides the possibility to easy update the I/O controller firmware. The driver must be started to use this feature. When the setup asks you for adding a new board or updating the driver and firmware type "No" for updating. A DOS box appears and reports about the state of the updating.

The error message

ERROR: \\.\COMxy -- The system cannot find the file specified.

in most cases signs that the driver isn't started. This may happen on WinNT systems when the driver start mode was modified to “Manual” or the device was disabled. The latter is possible on Windows 2000 systems also.

After the firmware was updated successfully you have to reboot the system to start the new firmware module on the I/O controller. The firmware actual running is executed from RAM.

The utility used by the driver setup tool to update the firmware is the Win32 Console Application “comtest.exe”, that is called with the following arguments from a command window:

comtest -ff=<FirmwareModule> <dev>

where <FirmwareModule> is the name of the file containing the firmware module to download and <dev> is the name of a port on the corresponding I/O controller (e.g. COM11). This allows the aimed firmware update on a particular controller.

The call of

comtest -?

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

gives a description of all options and arguments accepted by the utility. The application “comtest.exe” can be included into other applications or batch files when the possibility for automatic firmware update is desired.

Windows NT4/2000

The driver “ekf960si1" delivered by EKF was tested under Windows NT4 with at least service pack 4 installed. On a Windows 2000 system all driver tests were performed without a service pack installed.

Note, that the current version of the driver (1.30.0.0) does not support PnP on Windows 2000. The PCI address, i.e. PCI bus number and device number, must be entered under the driver key in the registry when installing the driver. See next section for details.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Installation

On a Windows NT4 system simply run the “setup.exe” installer on the installation disk labeled

"EKF's Intelligent Serial Driver for i960 Board Family WinNT/2000"

and follow the instructions. At the end of the installation procedure you should reboot the system to start the driver.

The driver currently distributed does not support Windows 2000's PnP. When installing a new board of EKF’s Intelligent I/O Controller Family to the system, the PnP manager of Windows 2000 reports that it has found new hardware when booting. Insert the installation disk in the floppy disk drive when Windows asks for a driver for the new hardware. After the PnP manager has done its work, it requested you to reboot the system. Before doing that call the “setup.exe” on the installation disk to configure anything necessary for the driver. Reboot the system when setup requested it.

Setup asks for the location on the *CompactPCI* bus, where the hardware to install can be found. The information needed by setup is the PCI bus number and the PCI device number. These numbers can be obtained in different ways:

- Take a look on the PCI device table listed by the system BIOS at boot time. Locate the I/O controller to install by the device type “Simple communication controller” or “Serial bus controller”.
- When the system is already booted use one of the several available PCI browsers like “PCIView”.
- Alternately call from a command window:

  pflash960 -Vm=100

  and you will get a PCI device listing. Look for devices with vendor ID 0x8086 and device ID 0x1960 to locate a member of EKF’s Intelligent I/O Controller family.

Pay attention to enter the PCI numbers in decimal when setup prompts for them.

Note, that when swapping an I/O controller to another *CompactPCI* slot, you have to change the board’s PCI numbers within the registry. See next section for the meaning of the registry entries used by the driver.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Registry Entries

Many parameters of the driver are controlled by registry entries. Normally there is no need to change these keys manually because the setup procedure will do all the work for you. Caution is given if you make any changes to the entries. Note down the old value before modifying an entry, so you can restore it if you run into trouble.

The following section gives an explanation of all registry keys the driver will use. After you have changed anything you should stop and then restart the driver by using the *Devices Menu* in the *Control Panel* (Windows NT4 only) or reboot the system.

Driver Basic Entry

The name of the main driver key for the driver “ekf960si1" is

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\ekf960si1

A few of the entries in the main driver key are common to all Windows NT4/2000 drivers. Note that all values shown are hexadecimal values.

"**ErrorControl**"=dword:00000001	log errors and display a message box.
"**Type**"=dword:00000001	defines driver as kernel-mode driver.
"**Start**"=dword:00000002	automatic driver start on system boot.

The next ones are driver specific:

"**RxFIFO**"=dword:00000008	receive FIFO high watermark, allowed values are 1, 4, 8, 14.
"**TxFIFO**"=dword:00000010	transmit FIFO size, allowed values are 1-16.
"**DebugLevel**"=dword:C000001F	debugging level, only useful for the checked version of the driver.

Board Subkeys

For each installed I/O controller exists one “board_x” subkey, where x is a one based, running index. Board subkeys must be consecutive. The board subkey contains the following entries:

HKEY_LOCAL_MACHINE\...\ekf960si1\board_x

"**BOARD_NAME**"="CU1-CHORUS"	name of the installed board.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

"**DEVICES_PER_BOARD**"=dword:00000010
number of ports on the board,
e.g.: CHORUS: 16, QUARTET: 4.

"**PCI_BUS_NUMBER**"=dword:00000001
number of the PCI bus where the board is attached to.

"**PCI_BUS_SLOT**"=dword:0000000B number of the PCI device where the board is attached to. Do not confuse this number with the *CompactPCI* rack slot number.

"**VALID**"=dword:00000001 defines wether the board should be ignored (VALID = 0) by the driver.

"**UART_CLOCK**"=dword:00E10000 UART clock frequency in Hz, depends on the oscillator mounted on the board, allowed is:
3686400, 8000000,
14745600, 16000000.

Parameter\channel Subkeys

For each port one the “board_x” exists one “channel_i” subkey, where i is the one based, running port index. The channel subkey is located under the Parameters subkey and contains the following entries:

HKEY_LOCAL_MACHINE\...\ekf960si1\board_x\Parameters\channel_i

"**COM_PORT_NUMBER**"=dword:0000000B
number of the COM port attached to this port, e.g. COM11.

"**NT_PORT_NUMBER**"=dword:0000000B
number of the NT device name attached to this port, e.g. Serial11.

"**DISABLE_PORT**"=dword:00000000 defines wether the port should be disabled (DISABLE_PORT != 0) by the driver.

Installations for the CX1-BAND CANbus controller contain also these entries for the CANbus ports, i.e. channel_2 and channel_3 (channel_1 is a normal serial port):

"**ACCEPTANCE_CODE**"=dword:00000000
acceptance code to build the CANbus controllers acceptance filter. [1])

"**ACCEPTANCE_MASK**"=dword:FFFFFFFF
acceptance mask to build the CANbus controllers acceptance filter. [1])

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

"**ACCEPTANCE_SINGLE**"=dword:00000001	build a single acceptance filter when set (ACCEPTANCE_SINGLE != 0), else a dual filter configuration is created. [1])
"**BASE_CLOCK**"=dword:016E3600	CANbus clock frequency in Hz, depends on the oscillator mounted on the board, allowed values are: 8000000, 16000000, 24000000.
"**DEVICE_TYPE**"=dword:00000001	specifies the type of the I/O device: 0 or absent defines a serial port, 1 defines a CANbus port.

Note:
[1]) The default acceptance filter created by the driver setup for all CANbus ports is configured so that the port accepts any frame. See the SJA1000 Data Sheet for details how to create an acceptance filter.

Restart The Device Driver

After any changes have been made to the registry entries a restart of the driver is necessary. This is best done via the *Devices Menu* of the *Control Panel* (Windows NT4 only). First stop the driver and then start it again.

On Windows 2000 reboot the system to take the changes affect.

Look in the event list to check that your changes haven't done any strange things.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Driver Dispatch Functions

The driver supports most of the functionality of the usual Windows NT4/2000 “Serial” device driver for the serial ports. Standard applications like the “HyperTerminal” are working with the serial ports on EKF’s Intelligent I/O Controllers without any problem.

When access to serial ports from a command window is required it is to note, that only the four ports COM1, COM2, COM3 and COM4 are supported by a command shell. Since the setup procedure configures port numbers on EKF’s Intelligent I/O Controllers starting with COM11, a renaming of the port’s COM_PORT_NUMBER under the driver registry key is necessary. See section “**Registry Entries**” for details. Note also, that on many systems COM1 and COM2 already used for the standard PC serial ports.

The CANbus ports are accessed in the same manner like their serial companions as usual COM ports, although because of their physical nature some limitations are existent. In most cases CANbus ports are embedded in proprietary applications written by the user. The static library “ekf960si1.lib” delivered with the device driver pack (C sources included) contains basic routines to read or write data via a CANbus port.

Data Structures Used By The Driver

The data structures used by the driver are explained in the following sections. They and their possibly corresponding definitions can be found in the C header “ntddekf.h” delivered with the driver installation pack.

SERIAL_CHARS

A structure that contains special characters used for serial ports:

```
typedef struct _SERIAL_CHARS
{
        UCHAR EofChar;
        UCHAR ErrorChar;
        UCHAR BreakChar;
        UCHAR EventChar;
        UCHAR XonChar;
        UCHAR XoffChar;
} SERIAL_CHARS, *PSERIAL_CHARS;
```

EofChar:
End Of File character (currently not used).

ErrorChar:
This character, when enabled, is placed in the stream of received characters on error conditions like buffer overflow, frame errors and so on.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

BreakChar:
This character, when enabled, is placed in the stream of received characters when a break condition was detected.

EventChar:
When enabled, an event is sent by the driver to the application, if this character was received by the port.

XonChar:
Defines the XON character that resumes an earlier stopped data transmission if XON/XOFF flow control is enabled.

XoffChar:
Defines the XOFF character that stops data transmission if XON/XOFF flow control is enabled.

SERIAL_HANDFLOW

A structure that contains all the stuff needed to setup hard- and software handshake for serial ports:

```
typedef struct _SERIAL_HANDFLOW
{
        ULONG ControlHandShake;
        ULONG FlowReplace;
        LONG XonLimit;
        LONG XoffLimit;
} SERIAL_HANDFLOW, *PSERIAL_HANDFLOW;
```

ControlHandShake:
A set of flags that defines the modem lines that are used for flow control:
SERIAL_DTR_HANDSHAKE:
Use the modem signal DTR for input flow control. The DTR line is cleared by the controller if the receive buffer reaches the programmed high water mark. See also description of XonLimit and XoffLimit.
SERIAL_CTS_HANDSHAKE:
SERIAL_DCD_HANDSHAKE:
SERIAL_DSR_HANDSHAKE:
Use the modem signal CTS, DCD or DSR respectively for output flow control. If the corresponding modem line(s) found as cleared, the controller will hold data transmission.
SERIAL_DSR_SENSITIVITY:
Ignore any character arriving when the DSR line is not set.
SERIAL_ERROR_ABORT:
If there exists an error condition the driver abort all read and writes to or from this port.

FlowReplace:
A set of flags defining flow control stuff:

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

SERIAL_AUTO_TRANSMIT:
Use the XON/XOFF protocol based flow control for output. The reception of the XoffChar will stop data transmission until the XonChar is received (see also structure SERIAL_CHARS).

SERIAL_AUTO_RECEIVE:
Use the XON/XOFF protocol based flow control for input. The XoffChar is send by the controller if the receive buffer reaches the programmed high water mark. If the receive buffer falls below the programmed low water mark, the XonChar is send. See also description of XonLimit and XoffLimit and of structure SERIAL_CHARS.

SERIAL_ERROR_CHAR:
If set, the ErrorChar is placed in the stream of received characters on error conditions like buffer overflow, frame errors and so on. See also description of structure SERIAL_CHARS.

SERIAL_NULL_STRIPPING:
If set, the reception of a NULL character is ignored.

SERIAL_BREAK_CHAR:
If set, the BreakChar is placed in the stream of received characters when a break condition was detected. See also description of structure SERIAL_CHARS.

SERIAL_RTS_HANDSHAKE:
Use the modem signal RTS for input flow control. The RTS line is cleared by the controller if the receive buffer reaches the programmed high water mark. See also description of XonLimit and XoffLimit.

XonLimit:
When there are less than XonLimit number of characters in the read buffer the controller will perform all flow control that the host has enabled so that the sender will resume sending characters.

XoffLimit:
When there are more characters than (BufferSize - XoffLimit) in the read buffer then the controller will perform all flow control that the host has enabled so that the sender will stop sending characters.

SERIALPERF_STATS

A structure that is used to get the current performance statistic counter values of a serial port.

```
typedef struct _SERIALPERF_STATS
{
    ULONG ReceivedCount;
    ULONG TransmittedCount;
    ULONG FrameErrorCount;
    ULONG SerialOverrunErrorCount;
    ULONG BufferOverrunErrorCount;
    ULONG ParityErrorCount;
} SERIALPERF_STATS, *PSERIALPERF_STATS;
```


EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

ReceivedCount:
The number of characters received successfully.

TransmittedCount:
The number of characters transmitted successfully.

FrameErrorCount:
The number of framing errors detected by the serial controller.

SerialOverrunErrorCount:
The number of overruns of the serial controller's internal receive FIFO.

BufferOverrunErrorCount:
The number of overruns of the read ring buffer maintained by the firmware.

ParityErrorCount:
The number of parity errors detected by the serial controller.

SERIAL_STATUS

A structure that is used to get the current error and general status of a serial port.

```
typedef struct _SERIAL_STATUS
{
        ULONG Errors;
        ULONG HoldReasons;
        ULONG AmountInInQueue;
        ULONG AmountInOutQueue;
        BOOLEAN EofReceived;
        BOOLEAN WaitForImmediate;
} SERIAL_STATUS, *PSERIAL_STATUS;
```

Errors:
A set of flags that reflect the possible errors occurred on a serial port:
SERIAL_ERROR_BREAK: a break condition was detected,
SERIAL_ERROR_FRAMING: a framing error was detected,
SERIAL_ERROR_OVERRUN: an overrun of the serial controller's internal receiver FIFO occurred,
SERIAL_ERROR_BUFFEROVERRUN: an overrun of the read ring buffer maintained by the firmware occurred,
SERIAL_ERROR_PARITY: a parity error was detected.

HoldReasons:
A set of flags that reflects the reasons why a port could be holding:
SERIAL_TX_WAITING_FOR_CTS
SERIAL_TX_WAITING_FOR_DSR
SERIAL_TX_WAITING_FOR_DCD
SERIAL_TX_WAITING_FOR_XON
SERIAL_TX_WAITING_XOFF_SENT

SERIAL_TX_WAITING_ON_BREAK
SERIAL_RX_WAITING_FOR_DSR

AmountInInQueue:
The number of bytes that reside currently in the port's read ring buffer.

AmountInOutQueue:
The number of bytes that reside currently in the port's write buffer.

EofReceived,
WaitForImmediate:
These flags are not used by the driver and will always return FALSE.

SERIAL_TIMEOUTS

A structure that contains all the stuff needed to setup timeouts on read or write requests for serial or CANbus ports:

```
typedef struct _SERIAL_TIMEOUTS
{
    ULONG ReadIntervalTimeout;
    ULONG ReadTotalTimeoutMultiplier;
    ULONG ReadTotalTimeoutConstant;
    ULONG WriteTotalTimeoutMultiplier;
    ULONG WriteTotalTimeoutConstant;
} SERIAL_TIMEOUTS, *PSERIAL_TIMEOUTS;
```

ReadIntervalTimeout:
The maximum time in milliseconds that may elapse between the reception of two characters. This kind of timeout is not supported for CANbus devices.

ReadTotalTimeoutMultiplier:
This time in milliseconds is multiplied by the number of characters that the current read wants to get. The total read timeout results of this product plus the value of ReadTotalTimeoutConstant.

ReadTotalTimeoutConstant:
This time in milliseconds is added to the product of the number of characters that the current read wants to get and the value of ReadTotalTimeoutMultiplier.

WriteTotalTimeoutMultiplier:
This time in milliseconds is multiplied by the number of characters that reside in the write buffer when the write was requested. The total write timeout results of this product plus the value of WriteTotalTimeoutConstant.

WriteTotalTimeoutConstant:
This time in milliseconds is added to the product of the number of characters in the write buffer and the value of WriteTotalTimeoutMultiplier.

Loading / Unloading The Driver

At boot time the system will automatically load and start the driver. The driver then looks in the registry for the installed hardware it supports, initializes all ports and makes them visible to the system.

The driver also supports its unload from the system. This can be made under Windows NT4 from the *Devices Menu* of the *Control Panel*. The devices administrated by the driver will be disabled, all allocated resources like memory and interrupt vectors are returned to the system. Last the devices will be deleted from the system making them invisible to any application.

Opening / Closing Of Devices

An application opens a port on an EKF Intelligent I/O Controller using the C function *CreateFile*. An individual port is referenced by its WIN32 device name, e.g. COM11. Opening a device for overlapped operation is always possible.

On device opening the driver purges the port's read and write buffers and clears the performance counters and the event mask. For serial devices additional the escape character is cleared and the thresholds for the flow control are set to XoffLimit = 511 and XonLimit = 2046.

If the port was opened the first time after system boot, the port has, dependent on its type, the following default parameters:

Serial ports: 9600 Baud, 8 data bits, 1 stop bit, no parity, the valid data mask is 0xFF, XOFF character is 0x13 (CTRL S), XON character is 0x11 (CTRL Q), any flow control is turned off, all timeout timers are turned off.

CANbus ports: 250 kBaud, acceptance filter as set up in the driver's registry key, all timeout timers are turned off.

Note, that if these device parameters are modified by IoControls, their values will remain across opens. They will never return to their initial values as found on the first open.

Example:
```
handle = CreateFile(
            "\\\\.\\COM11",
            GENERIC_READ | GENERIC_WRITE,
            0,
            NULL,
            OPEN_EXISTING,
            FILE_FLAG_OVERLAPPED,
            NULL
);
```


EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

CreateFile returns a handle to the device opened. If the opening failed, the constant INVALID_HANDLE_VALUE is returned and a call to *GetLastError* returns a corresponding error code.

To close an previously opened port the C function *CloseHandle* should be used. When closing a serial port, the driver waits for the transmission of the data that currently reside in the transmit FIFO of the UART. If programmed for XON/XOFF flow control, an XON character is sent when the reception was held before by sending XOFF. After that the driver waits 10 character times before clearing the DTR and RTS lines.

```
Call:   CloseHandle(
            handle              // handle returned by CreateFile
        );
```

CloseHandle returns TRUE on success. If the closing of the device failed, FALSE is returned and a call to *GetLastError* returns a corresponding error code.

Write Data

Data transmission via a previously successfully opened port is provided by the C function *WriteFile*. The data buffer given to *WriteFile* is sent unchanged to the port.

```
Call:   WriteFile(
            handle,             // handle returned by CreateFile
            pBuffer,            // pointer to data buffer
            bytesToWrite,       // number of bytes to write
            pBytesWritten,      // pointer to number of bytes written
            pOverlapped         // pointer to overlapped buffer
        );
```

WriteFile returns TRUE on success. If the write operation failed, FALSE is returned and a call to *GetLastError* returns a corresponding error code.

If the port was opened with FILE_FLAG_OVERLAPPED for asynchronous I/O, *WriteFile* returns FALSE and *GetLastError* may return ERROR_IO_PENDING. In that case a subsequent call to *GetOverlappedResult* is necessary. See also the Windows “Visual C++” documentation for details of asynchronous I/O operations.

On CANbus ports a complete frame including frame information field, transmit identifier and transmit data must be supplied to *WriteFile*. The driver will return an error, if a bad frame was passed. Therefore it is better to use the C function *Ekf960SendCanFrame* within the static library “ekf960si1.lib” to send frames over a CANbus port. This routine sets up a frame from the data supplied by the user and then calls *WriteFile* and, if necessary, *GetOverlappedResult*. See also the source file of *Ekf960SendCanFrame* “sendcan.c” that is delivered with the driver installation package.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

```
Call:   Ekf960SendCanFrame(
            handle,             // handle returned by CreateFile
            pOverlapped,        // optional  pointer to overlapped buffer (may be NULL)
            SendID,             // transmit identifier
            CAN_FLAG_EXTENDED,
                                // send an extended (29 bit ID) frame
            bytesToWrite,       // number of data bytes to write
            pBytesWritten,      // pointer to number of data bytes written
            pData,              // pointer to data buffer
            pStatusRecord       // optional pointer to status record (may be NULL)
        );
```

The write operation can be timed-out by setting up a timer with the I/O control request IOCTL_SERIAL_SET_TIMEOUTS.

Read Data

Data reception via a previously successfully opened port is provided by the C function Read*File*. The data received by the port is written unchanged to the buffer passed to the function if the following is true:

- no XON/XOFF flow control is used,
- null stripping mode is turned off,
- the error and break character insertion is turned off,
- insertion mode is turned off.

```
Call:   ReadFile(
            handle,             // handle returned by CreateFile
            pBuffer,            // pointer to data buffer
            bytesToRead,        // number of bytes to read
            pBytesRead,         // pointer to number of bytes read
            pOverlapped         // pointer to overlapped buffer
        );
```

ReadFile returns TRUE on success. If the read operation failed, FALSE is returned and a call to *GetLastError* returns a corresponding error code.

If the port was opened with FILE_FLAG_OVERLAPPED for asynchronous I/O *ReadFile* returns FALSE and *GetLastError* may return ERROR_IO_PENDING. In that case a subsequent call to *GetOverlappedResult* is necessary. See also the Windows “Visual C++” documentation for details of asynchronous I/O operations.

On CANbus ports a complete frame including frame information field, received frame identifier and frame data is returned by *ReadFile*. Therefore it is better to use the C function *Ekf960ReceiveCanFrame* within the static library “ekf960si1.lib” to receive frames via a CANbus port. This routine calls *ReadFile* and, if necessary, *GetOverlappedResult* and splits the parts of the received frame. See also the source file of *Ekf960ReceiveCanFrame* “receivecan.c” that is delivered with the driver installation package.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

```
Call:   Ekf960ReceiveCanFrame(
            handle,             // handle returned by CreateFile
            pOverlapped,        // optional  pointer to overlapped buffer (may be NULL)
            pExtended,          // pointer to a boolean that will be TRUE if the received
                                // frame has an extended (29 bit) identifier
            pRemoteXmit,        // pointer to a boolean that will be TRUE if the received
                                // frame has the remote transmit request flag set
            pIdentifier,        // pointer to a long that will be filled with the identifier of
                                // the received frame
            pDataSize,          // pointer to number of data bytes received with the frame
            pData,              // pointer to data buffer
            pStatusRecord       // optional pointer to status record (may be NULL)
        );
```

The read operation can be timed-out by setting up a timer with the I/O control request IOCTL_SERIAL_SET_TIMEOUTS.

Cancel I/O

All pending read, write and I/O control requests pending can be cancelled by calling the cancel I/O dispatch entry of the driver. A user application does this by the C function *CancelIo*:

```
Call:   CancelIo(
            handle              // handle returned by CreateFile
        );
```

CancelIo returns TRUE on success. If the cancel operation failed, FALSE is returned and a call to *GetLastError* returns a corresponding error code.

I/O Control Requests

The device driver for EKF's Intelligent I/O Controllers supports many I/O control function entries to setup several device parameters or to obtain the current device status. The following table gives an overview about the I/O control requests supported by the device driver "ekf960si1" for different device types:

I/O Control Requests Supported

I/O Control Request	Serial Device	CANbus Device
IOCTL_EKF960SI1_FLASH_ERASE	✓	✓
IOCTL_EKF960SI1_FLASH_LOCK	✓	✓
IOCTL_EKF960SI1_FLASH_UNLOCK	✓	✓
IOCTL_EKF960SI1_FLASH_WRITE	✓	✓
IOCTL_EKF960SI1_GET_ACCEPTANCE		✓

I/O Control Request	Serial Device	CANbus Device
IOCTL_EKF960SI1_GET_STATS_CAN		✓
IOCTL_EKF960SI1_SET_ACCEPTANCE		✓
IOCTL_SERIAL_CLEAR_STATS	✓	✓
IOCTL_SERIAL_CLEAR_DTR	✓	
IOCTL_SERIAL_CLEAR_RTS	✓	
IOCTL_SERIAL_CONFIG_SIZE	✓	✓
IOCTL_SERIAL_GET_BAUD_RATE	✓	✓
IOCTL_SERIAL_GET_CHARS	✓	
IOCTL_SERIAL_GET_COMMSTATUS	✓	✓
IOCTL_SERIAL_GET_DTRRTS	✓	
IOCTL_SERIAL_GET_HANDFLOW	✓	
IOCTL_SERIAL_GET_LINE_CONTROL	✓	
IOCTL_SERIAL_GET_MODEMSTATUS	✓	
IOCTL_SERIAL_GET_PROPERTIES	✓	✓
IOCTL_SERIAL_GET_STATS	✓	
IOCTL_SERIAL_GET_TIMEOUTS	✓	✓
IOCTL_SERIAL_GET_WAIT_MASK	✓	✓
IOCTL_SERIAL_LSRMST_INSERT	✓	
IOCTL_SERIAL_PURGE	✓	✓
IOCTL_SERIAL_RESET_DEVICE	✓	✓
IOCTL_SERIAL_SET_BAUD_RATE	✓	✓
IOCTL_SERIAL_SET_BREAK_OFF	✓	
IOCTL_SERIAL_SET_BREAK_ON	✓	
IOCTL_SERIAL_SET_CHARS	✓	
IOCTL_SERIAL_SET_DTR	✓	
IOCTL_SERIAL_SET_HANDFLOW	✓	
IOCTL_SERIAL_SET_LINE_CONTROL	✓	
IOCTL_SERIAL_SET_QUEUE_SIZE	✓	✓
IOCTL_SERIAL_SET_RTS	✓	
IOCTL_SERIAL_SET_TIMEOUTS	✓	✓
IOCTL_SERIAL_SET_WAIT_MASK	✓	✓
IOCTL_SERIAL_WAIT_ON_MAKS	✓	✓

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Most of these I/O controls are entered by the C function *DeviceIoControl*, nevertheless exists a couple of specialized C functions. The general format of DeviceIoControl is:

```
BOOL DeviceIoControl(
        HANDLE hDevice,                 // handle to device
        DWORD dwIoControlCode,          // operation
        PVOID lpInBuffer,               // input data buffer
        DWORD nInBufferSize,            // size of input data buffer
        PVOID lpOutBuffer,              // output data buffer
        DWORD nOutBufferSize,           // size of output data buffer
        PDWORD lpBytesReturned,         // byte count
        POVERLAPPED lpOverlapped        // overlapped information
);
```

DeviceIoControl returns TRUE on success. If the I/O control request failed, FALSE is returned and a call to *GetLastError* returns a corresponding error code.

The following sections will give a description of the I/O control requests supported by the “ekf960si1" driver in alphabetical order.

Erase Firmware Flash ROMs: IOCTL_EKF960SI1_FLASH_ERASE

This I/O control request erases the firmware flash ROMs on an I/O controller.

```
Call:   DeviceIoControl(
                handle,             // handle returned by CreateFile
                IOCTL_EKF960SI1_FLASH_ERASE,
                NULL,
                0,
                NULL,
                0,
                &unused,            // pointer to a DWORD variable
                pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Caution: Absolutely caution should be given when executing this I/O control request. Use makes sense only if a new firmware binary is available that is downloaded to the firmware flash ROMs with the I/O control request IOCTL_EKF960SI1_FLASH_WRITE after erasing. **The board is no more functional, if a hardware reset occurred while or after erasing.**

The application may open any port that resides on the corresponding board when calling the flash ROM erasure request. A call of IOCTL_EKF960SI1_LOCK should be done before calling the erase request to avoid confusion, if several threads try to update the same board firmware.

Lock Firmware Flash ROMs: IOCTL_EKF960SI1_FLASH_LOCK

This I/O control request locks the firmware flash ROMs on an I/O controller for exclusive use.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_EKF960SI1_FLASH_LOCK,
            NULL,
            0,
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

The application may open any port that resides on the corresponding board when calling the flash ROM lock request. A call of this request should be done before calling any other of the flash worker routines like erasure or writing to avoid confusion, if several threads try to update the same board firmware. After the complete firmware update is done a call to IOCTL_EKF960SI1_FLASH_UNLOCK is necessary.

Unlock Firmware Flash ROMs: IOCTL_EKF960SI1_FLASH_UNLOCK

This I/O control request unlocks the firmware flash ROMs on an I/O controller previously locked by IOCTL_EKF960SI1_FLASH_LOCK.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_EKF960SI1_FLASH_UNLOCK,
            NULL,
            0,
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Write Firmware Flash ROMs: IOCTL_EKF960SI1_FLASH_WRITE

This I/O control request writes a block of data to the flash ROMs of an I/O controller.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_EKF960SI1_FLASH_WRITE,
            pData,              // pointer to a EKF_DOWNLOAD_PARAMS structure [1])
            sizeof(*pData) + pData->byteCount,
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) This structure contains the size of the download data block and the offset within the flash ROMs where to write it. The download data is placed directly behind the parameter structure. It is defined in the C header file “ntddekf.h” delivered with the driver installation package. See also section “**EKF_DOWNLOAD_PARAMS**” in chapter “**Board Level Interface Description**”.

Caution: Absolutely caution should be given when executing this I/O control request. Use makes sense only if a new firmware binary is available that is downloaded to the firmware flash ROMs. Erasure of the old firmware with the I/O control request IOCTL_EKF960SI1_FLASH_ERASE is necessary before. **The board is no more functional, if a hardware reset occurred while writing the new firmware binary.**

The application may open any port that resides on the corresponding board when calling the flash ROM write request. A call of IOCTL_EKF960SI1_ERASE should be done before calling the write request.

The size of the download data block is limited to 16384 bytes (16KB) minus the size of the structure EKF_DOWNLOAD_PARAMS. Therefore it is necessary to split and download the firmware in several blocks.

After all blocks of the new firmware binary are written to the flash ROMs a board hardware reset must be supplied to start the new firmware.

Get Acceptance Filter: IOCTL_EKF960SI1_GET_ACCEPTANCE

This I/O control request returns the current acceptance filter setting of a CANbus port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_EKF960SI1_GET_ACCEPTANCE,
            NULL,
            0,
            pAcceptance,        // pointer to a SJA1000_ACCEPTANCE structure [1])
            sizeof(*pAcceptance),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) The acceptance filter setting is returned in this structure. It is defined in the C header file "ntddekf.h" delivered with the driver installation package.

An alternative way to get the acceptance filter is to use the C function *Ekf960GetAcceptance* coming with the library "ekf960si1.lib".

See also the source file of *Ekf960GetAcceptance* "getaccept.c" that is delivered with the driver installation package, the description of the I/O control request IOCTL_EKF960SI1_SET_ACCEPTANCE and the section "**SJA1000_ACCEPTANCE**" in chapter "**Board Level Interface Description**".

Get Performance Statistics: IOCTL_EKF960SI1_GET_STATS_CAN

This I/O control request returns the current performance statistics of a CANbus port. To get the statistics for a serial port use IOCTL_SERIAL_GET_STATS.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_EKF960SI1_GET_STATS_CAN,
            NULL,
            0,
            pStatistics,        // pointer to a SJA1000PERF_STATS structure [1])
            sizeof(*pStatistics),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) The performance statistics are returned in this structure. It is defined in the C header file "ntddekf.h" delivered with the driver installation package. See also the description in section "**SJA1000_PERF_STATS**" in the chapter "**Board Level Interface Description**" for an explanation of the fields of SJA1000PERF_STATS.

An alternative way to get the performance statistics is to use the C function *Ekf960GetStatisticsCan* coming with the library "ekf960si1.lib". See also the source file of *Ekf960GetStatisticsCan* "getstatscan.c" that is delivered with the driver installation package.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Setup Acceptance Filter: IOCTL_EKF960SI1_SET_ACCEPTANCE

This I/O control request sets up the acceptance filter for a CANbus port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_EKF960SI1_SET_ACCEPTANCE,
            pAcceptance,        // pointer to a SJA1000_ACCEPTANCE structure [1])
            sizeof(*pAcceptance),
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) This structure contains the acceptance filter settings. It is defined in the C header file "ntddekf.h" delivered with the driver installation package.

An alternative way to setup the acceptance filter is to use the C function *Ekf960SetAcceptance* coming with the library "ekf960si1.lib".

See also the source file of *Ekf960SetAcceptance* "setaccept.c" that is delivered with the driver installation package, the description of the I/O control request IOCTL_EKF960SI1_GET_ACCEPTANCE and the section "**SJA1000_ACCEPTANCE**" in chapter "**Board Level Interface Description**".

Clear Performance Statistics: IOCTL_SERIAL_CLEAR_STATS

This I/O control request clears the performance statistic counters for a serial or CANbus port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_CLEAR_STATS,
            NULL,
            0,
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```


EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Clear Modem Line DTR: IOCTL_SERIAL_CLEAR_DTR

This I/O control request clears the modem line *Data Terminal Ready* (DTR) on a serial port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_CLEAR_DTR,
            NULL,
            0,
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Clear Modem Line RTS: IOCTL_SERIAL_CLEAR_RTS

This I/O control request clears the modem line *Clear To Send* (RTS) on a serial port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_CLEAR_RTS,
            NULL,
            0,
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Get Configuration Size: IOCTL_SERIAL_CONFIG_SIZE

This I/O control request returns informations about the configuration size. The driver always returns 0 in the variable returnValue to this obsolete request.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_CONFIG_SIZE,
            NULL,
            0,
            &returnValue,       // pointer to a DWORD variable
            sizeof(returnValue),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```


EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Port Baud Rate: IOCTL_SERIAL_GET_BAUD_RATE

This I/O control request returns the baud rate that is currently set on a serial or CANbus port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_GET_BAUD_RATE,
            NULL,
            0,
            pBaudStruct,        // pointer to a SERIAL_BAUD_RATE structure [1])
            sizeof(*pBaudStruct),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) The baud rate value is returned in this structure. It is defined in the C header file "ntddekf.h" delivered with the driver installation package.

An alternative way to get the current baud rate is to use the C function *Ekf960GetBaudRate* coming with the library "ekf960si1.lib". See also the source file of *Ekf960GetBaudRate* "getbaud.c" that is delivered with the driver installation package.

Get Special Characters: IOCTL_SERIAL_GET_CHARS

This I/O control request returns the special characters (e.g. XON and XOFF characters) that are currently set on a serial port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_GET_CHARS,
            NULL,
            0,
            pChars,             // pointer to a SERIAL_CHARS structure [1])
            sizeof(*pChars),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) The special characters are returned in this structure. It is defined in the C header file "ntddekf.h" delivered with the driver installation package. See also description in section "**SERIAL_CHARS**"

Get Port Status: IOCTL_SERIAL_GET_COMMSTATUS

This I/O control request returns the current communication status of a serial or CANbus port. This includes the number of characters in the read and write buffers, the error status and so on.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_GET_COMMSTATUS,
            NULL,
            0,
            pStatus,            // pointer to a SERIAL_STATUS or SJA1000_STATUS
                                // structure [1])
            sizeof(*pStatus),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Notes:
[1]) The status is returned in these structures. Use SERIAL_STATUS for serial ports and SJA1000_STATUS for CANbus ports. These structures are defined in the C header file "ntddekf.h" delivered with the driver installation package. See also section "**SERIAL_STATUS**" in this chapter and section "**SJA1000_STATUS**" in the chapter "**Board Level Interface Description**".
[2]) The error status word kept by the driver and returned in the status record is cleared after this request was executed.

Get DTR/RTS Status: IOCTL_SERIAL_GET_DTRRTS

This I/O control request returns the current status of the modem lines *Data Terminal Ready* (DTR) and *Request To Send* (RTS) of a serial port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_GET_DTRRTS,
            NULL,
            0,
            &Status,            // pointer to a DWORD variable [1])
            sizeof(Status),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```


EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Note:
[1]) The DWORD Status contains zero or more of the following flags:

SERIAL_DTR_STATE: DTR line is set
SERIAL_RTS_STATE: RTS line is set

The flags are defined in the C header file “ntddekf.h” delivered with the driver installation package.

Get Flow Control: IOCTL_SERIAL_GET_HANDFLOW

This I/O control request returns the handshake and flow control that currently is set on a serial port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_GET_HANDFLOW,
            NULL,
            0,
            pHandFlow,          // pointer to a SERIAL_HANDFLOW structure [1])
            sizeof(*pHandFlow),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) The flow control settings are returned in this structure. It is defined in the C header file “ntddekf.h” delivered with the driver installation package. See also description in section “**SERIAL_HANDFLOW**”.

Get Line Control: IOCTL_SERIAL_GET_LINE_CONTROL

This I/O control request returns the current line control settings of a serial port. This includes the number of data bits, number of stop bits and the parity.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_GET_LINE_CONTROL,
            NULL,
            0,
            pLineControl,       // pointer to a SERIAL_LINE_CONTROL structure [1])
            sizeof(*pLineControl),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```


EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Note:
[1]) The line control settings are returned in this structure. It is defined in the C header file "ntddekf.h" delivered with the driver installation package.

Get Modem Status: IOCTL_SERIAL_GET_MODEMSTATUS

This I/O control request returns the current modem line settings of a serial port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_GET_MODEMSTATUS,
            NULL,
            0,
            &Status,            // pointer to a DWORD variable [1])
            sizeof(Status),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) The modem status settings are returned in this DWORD variable. The value returned reflects the current contents of the UART's modem status register (MSR). See UART 16550 data sheet for details of the MSR.

Get Device Properties: IOCTL_SERIAL_GET_PROPERTIES

This I/O control request returns information about the capabilities of a serial port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_GET_PROPERTIES,
            NULL,
            0,
            pProperties,        // pointer to a SERIAL_COMMPROP structure [1])
            sizeof(*pProperties),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) The capability information is returned in this structure. It is defined in the C header file "ntddekf.h" delivered with the driver installation package.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Performance Statistics: IOCTL_SERIAL_GET_STATS

This I/O control request returns the current performance statistics of a serial port. To get the statistics for a CANbus port use IOCTL_EKF960SI1_GET_STATS_CAN.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_GET_STATS,
            NULL,
            0,
            pStatistics,        // pointer to a SERIALPERF_STATS structure [1])
            sizeof(*pStatistics),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) The performance statistics are returned in this structure. It is defined in the C header file “ntddekf.h” delivered with the driver installation package.

Get Timeout Settings: IOCTL_SERIAL_GET_TIMEOUTS

This I/O control request returns the current timeout settings of a serial or CANbus port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_GET_TIMEOUTS,
            NULL,
            0,
            pTimeouts,          // pointer to a SERIAL_TIMEOUTS structure [1])
            sizeof(*pTimeouts),
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) The timeout parameters are returned in this structure. It is defined in the C header file “ntddekf.h” delivered with the driver installation package. See also description in section “**SERIAL_TIMEOUTS**”. The time base of all timeouts within SERIAL_TIMEOUTS is milliseconds.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Wait Mask Setting: IOCTL_SERIAL_GET_WAIT_MASK

This I/O control request returns the event mask that is currently set on a serial or CANbus port.

```
Call:   DeviceIoControl(
                handle,             // handle returned by CreateFile
                IOCTL_SERIAL_GET_WAIT_MASK,
                NULL,
                0,
                &WaitMask,          // pointer to a DWORD variable
                sizeof(WaitMask),
                &unused,            // pointer to a DWORD variable
                pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) The DWORD WaitMask contains a set of zero or more flags that indicate which events currently are enabled. The flags are defined in the C header file “ntddekf.h” delivered with the driver installation package. See I/O control request IOCTL_SERIAL_SET_WAIT_MASK for a description of these flags.

Setup Insert Mode: IOCTL_SERIAL_LSRMST_INSERT

This I/O control request is used to enable or disable the insertion of information about the line status and the modem status in the received data stream of a serial port. The information inserted always starts with the escape character passed with this I/O control request and a following character describing the type of event happened. Passing an escape character of 0x00 will disable the insertion mode.

On receive error the escape character and the character SERIAL_LSRMST_LSR_DATA are placed in the read buffer followed by the contents of the *Line Status Register* (LSR) of the serial controller and the character received.

If no receive data was available when the receive error occurred, the escape character, SERIAL_LSRMST_LSR_NODATA and the contents of LSR is placed in the read buffer.

On changes of the modem lines the escape character, SERIAL_LSRMST_MST and the contents of the *Modem Status Register* (MSR) is placed in the read buffer.

The reception of the escape character itself is indicated by the insertion of the sequence escape character, SERIAL_LSRMST_ESCAPE, escape character.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_LSRMST_INSERT,
            &EscapeChar,        // pointer to an UCHAR variable containing the escape
                                // character
            sizeof(EscapeChar),
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Purge Read/Write Queues: IOCTL_SERIAL_PURGE

This I/O control request is used to cancel the specified requests and to remove any data from the corresponding queues.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_PURGE,
            &PurgeMask,         // pointer to a DWORD variable containing the purge
                                // mask [1])
            sizeof(PurgeMask),
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) The purge mask contains one or more of the following flags:

SERIAL_PURGE_RXABORT:	Cancel current and purge all read requests
SERIAL_PURGE_RXCLEAR:	Purge the read buffer
SERIAL_PURGE_TXABORT:	Cancel current and purge all write requests
SERIAL_PURGE_TXCLEAR:	Purge the write buffer

The flags are defined in the C header file “ntddekf.h” delivered with the driver installation package.

Reset The Device: IOCTL_SERIAL_RESET_DEVICE

This I/O control request is used to reset the device controller of a serial or CANbus port.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_RESET_DEVICE,
            NULL,
            0,
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
The device driver for EKF's Intelligent I/O Controllers really does nothing when calling this request.

Setup Port Baud Rate: IOCTL_SERIAL_SET_BAUD_RATE

This I/O control request sets the baud rate on a serial or CANbus port. The driver accepts all well known baud rates as defined in the C header file "ntddekf.h" (SERIAL_BAUD_075 through SERIAL_BAUD_115200). Furthermore any baud rate that can be programmed with a resulting error of less than 1% can be chosen.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_SET_BAUD_RATE,
            pBaudStruct,        // pointer to a SERIAL_BAUD_RATE structure [1])
            sizeof(*pBaudStruct),
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) This structure contains the baud rate value. It is defined in the C header file "ntddekf.h" delivered with the driver installation package.

An alternative way to setup the baud rate is to use the C function *Ekf960SetBaudRate* coming with the library "ekf960si1.lib":

```
    Ekf960SetBaudRate(
            handle,             // handle returned by CreateFile
            pOverlapped,        // optional  pointer to overlapped buffer (may be NULL)
            BaudRate            // baud rate in bits per second
        );
```

See also the source file of *Ekf960SetBaudRate* "setbaud.c" that is delivered with the driver installation package.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Set Break Off: IOCTL_SERIAL_SET_BREAK_OFF

This I/O control request is used to turn off the break condition on a serial device.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_SET BREAK_OFF,
            NULL,
            0,
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Set Break On: IOCTL_SERIAL_SET_BREAK_ON

This I/O control request is used to turn on the break condition on a serial device.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_SET BREAK_ON,
            NULL,
            0,
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Setup Special Characters: IOCTL_SERIAL_SET_CHARS

This I/O control request sets up the special characters (e.g. XON and XOFF characters) on a serial port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_SET_CHARS,
            pChars,             // pointer to a SERIAL_CHARS structure [1])
            sizeof(*pChars),
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```


EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Note:
[1]) This structure is used to pass the new special character setting. It is defined in the C header file “ntddekf.h” delivered with the driver installation package. See also description in section “**SERIAL_CHARS**”

Set Modem Line DTR: IOCTL_SERIAL_SET_DTR

This I/O control request sets the modem line *Data Terminal Ready* (DTR) on a serial port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_SET_DTR,
            NULL,
            0,
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Setup Flow Control: IOCTL_SERIAL_SET_HANDFLOW

This I/O control request sets up the handshake and flow control values on a serial port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_SET_HANDFLOW,
            pHandFlow,          // pointer to a SERIAL_HANDFLOW structure [1])
            sizeof(*pHandFlow),
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) This structure is used to pass the new handshake and flow control settings. It is defined in the C header file “ntddekf.h” delivered with the driver installation package. See also description in section “**SERIAL_HANDFLOW**”.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Setup Line Control: IOCTL_SERIAL_SET_LINE_CONTROL

This I/O control request sets up the line control values on a serial port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_SET_LINE_CONTROL,
            pLineControl,       // pointer to a SERIAL_LINE_CONTROL structure [1])
            sizeof(*pLineControl),
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) This structure is used to pass the new line control settings. It is defined in the C header file "ntddekf.h" delivered with the driver installation package. This file also contains definitions for character sizes, number of stop bits and parity settings (SERIAL_DATABITS_5 through SERIAL_PARITY_SPACE).

Setup Receive Buffer Size: IOCTL_SERIAL_SET_QUEUE_SIZE

This I/O control request sets up the receive buffer size of a serial port. Since EKF's Intelligent I/O Controllers always work with a fixed receive buffer size, this call will do nothing, if the new requested size is less or equal the current size. If a larger buffer is desired, an error is returned.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_SET_QUEUE_SIZE,
            pQueueSize,         // pointer to a SERIAL_QUEUE_SIZE structure [1])
            sizeof(*pQueueSize),
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) This structure is used to pass the new queue size. It is defined in the C header file "ntddekf.h" delivered with the driver installation package.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Set Modem Line RTS: IOCTL_SERIAL_SET_RTS

This I/O control request sets the modem line *Clear To Send* (RTS) on a serial port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_SET_RTS,
            NULL,
            0,
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Setup Timeouts: IOCTL_SERIAL_SET_TIMEOUTS

This I/O control request sets up the timeout values for read and write requests on a serial or CANbus port.

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_SET_TIMEOUTS,
            pTimeouts,          // pointer to a SERIAL_TIMEOUTS structure [1])
            sizeof(*pTimeouts),
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) This structure is used to pass the new timeout values. It is defined in the C header file "ntddekf.h" delivered with the driver installation package. See also description in section "**SERIAL_TIMEOUTS**". The time base of all timeouts within SERIAL_TIMEOUTS is milliseconds.
[2]) Read interval timeouts for CANbus devices are not supported.

Setup Wait Event Mask: IOCTL_SERIAL_SET_WAIT_MASK

This I/O control request sets up the wait event mask on a serial or CANbus port. This configures the driver to notify an application after the occurrence of at least one of the enabled events. To wait for an event, an application may use the I/O control request IOCTL_SERIAL_WAIT_ON_MASK.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

```
Call:   DeviceIoControl(
            handle,             // handle returned by CreateFile
            IOCTL_SERIAL_SET_WAIT_MASK,
            &WaitMask,          // pointer to a DWORD variable [1])
            sizeof(WaitMask),
            NULL,
            0,
            &unused,            // pointer to a DWORD variable
            pOverlapped         // optional  pointer to overlapped buffer (may be NULL)
        );
```

Note:
[1]) The DWORD WaitMask contains zero or more of the following flags when called for a serial port:

SERIAL_EV_RXCHAR:	Any Character received
SERIAL_EV_RXFLAG:	Received certain character
SERIAL_EV_TXEMPTY:	Transmit Queue Empty
SERIAL_EV_CTS:	CTS changed state
SERIAL_EV_DSR:	DSR changed state
SERIAL_EV_RLSD:	DCD changed state
SERIAL_EV_BREAK:	BREAK received
SERIAL_EV_ERR:	Line status error occurred
SERIAL_EV_RING:	Ring signal detected
SERIAL_EV_RX80FULL:	Receive buffer is 80 percent full

CANbus devices support these event flags:

SJA1000_EV_RXFRAME:	A frame received
SJA1000_EV_BUSOFF:	A bus-off event occurred
SJA1000_EV_BUSON:	A bus-on event occurred
SJA1000_EV_ERR:	An error occurred

The flags are defined in the C header file “ntddekf.h” delivered with the driver installation package.

Wait For An Event: IOCTL_SERIAL_WAIT_ON_MASK

This I/O control request waits for the occurrence of one or more event previously enabled by the I/O control request SERIAL_SET_WAIT_MASK. The request returns immediately if a wait event already occurred when calling.

Call: DeviceIoControl(
handle, // handle returned by *CreateFile*
IOCTL_SERIAL_WAIT_ON_MASK,
NULL,
0,
&Event, // pointer to a DWORD variable [1])
sizeof(Event),
&unused, // pointer to a DWORD variable
pOverlapped // optional pointer to overlapped buffer (may be NULL)
);

Note:
[1]) The DWORD Event is used to return the events that were occurred. It contains one or more of the event flags defined in the C header file “ntddekf.h” delivered with the driver installation package. See also I/O control request IOCTL_SERIAL_SET_WAIT_MASK for a description of these flags.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Static Library Ekf960si1.lib

The driver installation pack comes with the static library “ekf960si1.lib”. This small library currently mainly contains C functions for CANbus port processing, but will be enlarged in the future.

The library currently consists of the following functions:

```
BOOL
Ekf960GetBaudRate(                  // Get current device baud rate
        HANDLE handle,
        LPOVERLAPPED overlap,
        PULONG pBaudRate
        );

BOOL
Ekf960GetStatisticsCan(             // Get performance statistics of a CANbus device
        HANDLE handle,
        LPOVERLAPPED overlap,
        PSJA1000PERF_STATS pStatistics
        );

BOOL
Ekf960GetStatusCan(                 // Get I/O status of a CANbus device
        HANDLE handle,
        LPOVERLAPPED overlap,
        PSJA1000_STATUS pStatus
        );

BOOL
Ekf960ReceiveCanFrame(              // Receive a CAN frame
        HANDLE handle,
        LPOVERLAPPED overlap,
        PBOOL pExtended,
        PBOOL pRemoteXmit,
        PULONG pIdentifier,
        PULONG pDataSize,
        PUCHAR pData,
        PSJA1000_STATUS pStatus
        );
```

```
BOOL
Ekf960SendCanFrame(                    // Send a CAN frame
        HANDLE handle,
        LPOVERLAPPED overlap,
        ULONG identifier,
        ULONG flags,
        ULONG dataSize,
        PULONG pDataSend,
        PUCHAR pData,
        PSJA1000_STATUS pStatus
        );

BOOL
Ekf960SetAcceptance(                   // Set CANbus device acceptance filter
        HANDLE handle,
        LPOVERLAPPED overlap,
        ULONG AcceptCodeId1,
        ULONG AcceptCodeId2,
        ULONG AcceptCodeData,
        ULONG AcceptMaskId1,
        ULONG AcceptMaskId2,
        ULONG AcceptMaskData,
        ULONG Flags
        );

BOOL
Ekf960SetBaudRate(                     // Set device baud rate
        HANDLE handle,
        LPOVERLAPPED overlap,
        ULONG BaudRate
        );
```

For a detailed description of the library functions see their source files that are delivered with the driver package. These sources can also be included directly or modified to users applications.

Board Level Interface Description

This chapter describes the low level interface to a board of the EKF Intelligent I/O Controller family. It is dedicated to those software developers that have to write a driver for such a board.

To support driver writers, EKF delivers C header files that contain most of the necessary stuff like structure and macro definitions.

Board Firmware

EKF's Intelligent I/O Controllers are based on Intel's i960® Rx processor, a powerful 32-bit I/O controller which incorporates a multi-function PCI interface. The PCI device consists of a PCI-to-PCI Bridge at PCI function 0 and an Address Translation Unit (ATU) at PCI function 1. The Messaging Unit (called MU) is a part of the ATU. The MU is the core of the I/O board's interface.

The on-board firmware boots up automatically after power up or board reset (factory setting of the configuration jumper field JCNF on the I/O boards). It first initializes the local devices like memory, I/O ports and the (Compact)PCI interface, i.e. the MU. During this time all configuration cycles from the PCI interface to the board will be retried.

After all local initializations are done, PCI configuration cycles will be accepted again. At this time the hosts BIOS is able to scan for the device and to initialize its PCI resources. Thus it can be assumed that the controller is ready to work as soon as PCI cycles to it are possible.

Messaging Unit

The messaging unit provides data transfers between the PCI system and the i960®RP. It uses interrupts to notify each system when new data arrives. The MU has four messaging mechanisms: Message Registers, Doorbell Registers, Circular Queues and Index Registers. Each allows a host processor or external PCI device and the i960®RP to communicate through message passing and interrupt generation.

The MU is part of the primary Address Translation Unit (ATU) that is shortly described in the following section.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Address Translation Unit

The i960 Rx processor contains two Address Translation Units, a primary and a secondary ATU. The primary ATU builds the data path between the *CompactPCI* bus and the local data bus, i.e. the local memory. The secondary ATU is not used on any member of EKF's Intelligent I/O Controller family, thus the focus is set on the primary ATU only, that simply is called ATU in the following.

From the host's view the ATU is seen as PCI function 1. As every PCI device the ATU implements its own configuration space. This space is 256 bytes in size, whereas the first 64 bytes must adhere to a predefined header format. The ATU is programmed on the *CompactPCI* interface via type 0 configuration commands to PCI function 1.

The following table shows the configuration header according the *PCI Local Bus Specification*, revision 2.1:

<table>
<tr><th colspan="4">Address Translation Unit Configuration Header</th><th>Address Offset</th></tr>
<tr><td colspan="2">ATU Device ID</td><td colspan="2">ATU Vendor ID</td><td>0x00</td></tr>
<tr><td colspan="2">ATU Primary Status</td><td colspan="2">ATU Primary Command</td><td>0x04</td></tr>
<tr><td>ATU Base Class</td><td>ATU Sub Class</td><td>ATU Prog IF</td><td>ATU Revision ID</td><td>0x08</td></tr>
<tr><td>BIST</td><td>Header Type</td><td>Latency Timer</td><td>Cacheline Size</td><td>0x0C</td></tr>
<tr><td colspan="4">Primary Inbound ATU Base Address PIABAR</td><td>0x10</td></tr>
<tr><td colspan="4" rowspan="6">Reserved</td><td>0x14</td></tr>
<tr><td>0x18</td></tr>
<tr><td>0x1C</td></tr>
<tr><td>0x20</td></tr>
<tr><td>0x24</td></tr>
<tr><td>0x28</td></tr>
<tr><td colspan="2">ATU Subsystem ID</td><td colspan="2">ATU Subsystem Vendor ID</td><td>0x2C</td></tr>
<tr><td colspan="4">Expansion ROM Base Address</td><td>0x30</td></tr>
<tr><td colspan="4" rowspan="2">Reserved</td><td>0x34</td></tr>
<tr><td>0x38</td></tr>
<tr><td>Max. Latency</td><td>Minimum Grant</td><td>Interrupt Pin</td><td>Interrupt Line</td><td>0x3C</td></tr>
</table>

The next table shows the i960®RP specific registers of the ATU configuration space.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

ATU Extended PCI Configuration Register Space		Address Offset
Primary Inbound ATU Limit Register PIALR		0x40
Primary Inbound ATU Translate Value Register PIATVR		0x44
Secondary Inbound ATU Base Address Register SIABAR		0x48
Secondary Inbound ATU Limit Register SIALR		0x4C
Secondary Inbound ATU Translate Value Register SIATVR		0x50
Primary Outbound Memory Window Value Register POMWVR		0x54
Reserved		0x58
Primary Outbound I/O Window Value Register POIOWVR		0x5C
Primary Outbound DAC Window Value Register PODWVR		0x60
Primary Outbound Upper 64-bit DAC Register POUDR		0x64
Secondary Outbound Memory Window Value Register SOMWVR		0x68
Secondary Outbound I/O Window Value Register SOIOWVR		0x6C
Reserved		0x70
Expansion ROM Base Address Register ERBAR		0x74
Expansion ROM Translate Value Register ERTVR		0x78
Reserved		0x7C
		0x80
		0x84
ATU Configuration Register ATUCR		0x88
Reserved		0x8C
Primary ATU Interrupt Status Register PATUISR		0x90
Secondary ATU Interrupt Status Register SATUISR		0x94
Secondary ATU Status Register	Secondary ATU Commond Register	0x98
Secondary Outbound DAC Window Value Register SODWVR		0x9C
Secondary Outbound Upper 64-bit DAC Register SOUDR		0xA0
Primary Outbound Configuration Cycle Address Register POCCAR		0xA4
Secondary Outbound Configuration Cycle Address Register SOCCAR		0xA8
Primary Outbound Configuration Cycle Data Register POCCDR		0xAC
Secondary Outbound Configuration Cycle Data Register SOCDCR		0xB0
Reserved		0xB4 - 0xFF

A few of the fields within the ATU's configuration space are written by the firmware at boot time. These are the subsystem ID, the subsystem vendor ID and the device class code (see table below), the primary inbound ATU limit register PIALR, that defines the size of the PCI

window occupied by the PCI device. Others are filled by the system controller's BIOS, e.g. the primary inbound ATU base address register PIABAR.

To identify a board the firmware writes the following subsystem IDs and class codes to the corresponding registers:

Subsystem and Subvendor IDs

Board	Subvendor ID	Subsystem ID	Base Class Code	Sub Class Code
CG1-RADIO	0xE4BF	0x1010	0x07 [1])	0x02 [3])
CU1-CHORUS	0xE4BF	0x1040	0x07 [1])	0x02 [3])
CU2-QUARTET	0xE4BF	0x1020	0x07 [1])	0x02 [3])
CX1-BAND	0xE4BF	0x3100	0x0C [2])	0x09 [4])

Notes:
[1]) Base Class Code for Simple Communication Controllers
[2]) Base Class Code for Serial Bus Controllers
[3]) Sub Class Code for Multiport Serial Controller
[4]) Sub Class Code for CANbus Controller

The Messaging Unit appears as a set of memory mapped registers, starting at the base address defined in the register PIABAR (offset 0x10 in configuration space of function 1). The MU consists of a fixed header defined by the i960 processor (first 4KByte) and a following part defined by the software interface.

Read and write accesses of byte, short or long size are possible although there are a few registers containing bits that are read only or read/clear (e.g. the interrupt status registers).

The following table shows the structure of the MU, the address offset shown is with respect to the base address. Registers that are meaningful for the software interface are marked light blue/italic:

Structure of the Messaging Unit Registers

MU Register Name	Member Name [1])	Address Offset
APIC Register Select Register	ARSR	0x0000
Reserved		0x0004
APIC Window Register	AWR	0x0008
Reserved		0x000C
Inbound Message Register 0	*IMR0*	*0x0010*
Inbound Message Register 1	IMR1	0x0014
Outbound Message Register 0	*OMR0*	*0x0018*
Outbound Message Register 1	OMR1	0x001C
Inbound Doorbell Register	IDR	0x0020

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

MU Register Name	Member Name [1])	Address Offset
Inbound Interrupt Status Register	IISR	0x0024
Inbound Interrupt Mask Register	IIMR	0x0028
Outbound Doorbell Register	ODR	0x002C
Outbound Interrupt Status Register	*OISR*	*0x0030*
Outbound Interrupt Mask Register	*OIMR*	*0x0034*
Reserved		0x0038
		0x003C
Inbound Queue Port	IQP	0x0040
Outbound Queue Port	OQP	0x0044
Reserved		0x0048
		0x004C
Index Register 0 : : Index Register 1003	IDXR[0] : : IDXR[4015]	0x0050 : : 0x0FFC
Inbound Parameter Buffer	*ParameterBuffer.InBound*	*0x1000*
Outbound Parameter Buffer	*ParameterBuffer.OutBound*	*0x1800*
Download Buffer	*DownloadBuffer*	*0x2000*
Write Buffer Device 0	*IoBuffer[0].Write*	*0x6000*
Read Buffer Device 0	*IoBuffer[0].Read*	*0x7000*
Write Buffer Device 1	*IoBuffer[1].Write*	*0x8000*
Read Buffer Device 1	*IoBuffer[1].Read*	*0x9000*
...	*...*	*[2])*

Notes:
[1]) This is the name of the member of the structure I960_MESSAGE_UNIT defined in “ekf960if.h”.
[2]) The number of IoBuffers depends on the number of devices on a board.

The MU is described in the header file "ekf960if.h" as a structure of type I960_MESSAGE_UNIT. In the following an explanation is given about the MU registers that are important for the software interface discussed here. Again, the relevant bits are marked light blue/italic.

The Inbound Message Register 0 (IMR0) is used to send messages to the controller. When IMR0 is written, an interrupt is requested on the controller.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Inbound Message Register 0 (IMR0)

Bit	Access	Description
31:00	*Read/Write*	*Inbound Message Word.*

The Outbound Message Register 0 (OMR0) is used to send messages from the controller to the host. When OMR0 is written by the controller firmware, an interrupt is requested on the *CompactPCI* bus. Interrupt generation can be disabled by setting bit 0 in the Outbound Interrupt Mask Register OIMR.

Outbound Message Register 0 (OMR0)

Bit	Access	Description
31:00	*Read/Write*	*Outbound Message Word generated by the controller.*

The Outbound Interrupt Status Register (OISR) records the status of the PCI interrupts generated by the MU. If the firmware has send a message to the host this is indicated by the fact that bit 0 of the OISR is set. Interrupt generation may be masked by setting the corresponding bits in the Outbound Interrupt Mask Register (OIMR). All other bits in OISR will never be set by the controller.

Outbound Interrupt Status Register (OISR)

Bit	Access	Description
31:08	Read Only	Reserved (Read as 0)
7	Read Only	PCI Doorbell Interrupt D.
6	Read Only	PCI Doorbell Interrupt C.
5	Read Only	PCI Doorbell Interrupt B.
4	Read Only	PCI Doorbell Interrupt A.
3	Read Only	Outbound Post Queue Interrupt.
2	Read Only	Outbound Doorbell Interrupt.
1	Read/Clear	Outbound Message 1 Interrupt.
0	*Read/Clear*	*Outbound Message 0 Interrupt - set by the MU when the OMR0 is written by the i960 processor. Clearing this by writing a one to it clears the interrupt request.*

The Outbound Interrupt Mask Register (OIMR) is used to mask undesirable interrupts that may be generated by the MU. Each bit set in OIMR will disable the corresponding interrupt source. Note, that this register defaults to all bits cleared after reset, thus all interrupts are enabled.

Outbound Interrupt MASK Register (OIMR)

Bit	Access	Description
31:08	Read Only	Reserved (Read as 0)
7	Read/Write	PCI Doorbell Interrupt D Mask.
6	Read Only	PCI Doorbell Interrupt C Mask.
5	Read Only	PCI Doorbell Interrupt B Mask.
4	Read Only	PCI Doorbell Interrupt A Mask.
3	Read Only	Outbound Post Queue Interrupt Mask.
2	Read Only	Outbound Doorbell Interrupt Mask.
1	Read/Clear	Outbound Message 1 Interrupt Mask.
0	*Read/Clear*	*Outbound Message 0 Interrupt Mask.* *0 - interrupt generation allowed* *1 - interrupt generation disabled*

Mailbox

Although the MU offers many possibilities to exchange data and to request servicing of a new message, EKF's I/O boards only use the Inbound Message Register 0 (IMR0) to send a message to the controller. In return the controller sends messages via the Outbound Message Register 0 (OMR0).

Writing a message to the IMR0 will request an interrupt on the controller that is serviced by the firmware. After any necessary actions involved with the message are done, the controller's firmware answers by writing a message to the OMR0. If enabled in the Outbound Interrupt Mask Register OIMR (by clearing bit 0 of OIMR), this will request an interrupt on the *CompactPCI* bus.

The Outbound Interrupt Status Register (OISR) will reflect the source of the MU interrupt by setting bit 0. The interrupt request is cleared in the interrupt service routine by writing a one to this bit position (i.e. setting bit 0 of OISR).

Of course it is also possible to poll bit 0 of OISR to recognize a message from the controller if interrupt processing should be avoided.

Buffer / Parameter Areas

Messages to or from the controller may require parameters or pass data with them. Therefore data and parameter buffers are existing within the MU.

Parameters are passed to the controller by the inbound parameter buffer while in the opposite direction the outbound parameter buffer is used. There exists one inbound and one outbound parameter buffer each of fixed size that are shared by all devices on the I/O

controller. The inbound parameter buffer is used to pass parameters belonging to a message to the controller. The controller returns parameters in the outbound parameter area as a result of a previously sent message.

Write or read data are exchanged in the I/O buffer areas. Each port owns one read and one write buffer each of fixed size. A buffer consists of a field that contains the current amount of data in the buffer and the buffer itself. The buffer structure is described in the struct EKF_MU_IO_BUFFER in the C header file "ekf960if.h".

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Command Word Structure

Commands passed to the controller via the Inbound Message Register IMR0 are called messages. A message consists of a command word and belonging parameters or data. The command word contains the meaning of the message, the device number involved with the message, the size of belonging parameters or data and some flags. The next figure shows the general layout of the command word, 32 bit in size:

General Command Word Format

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
31			28	27			24	23							16	15							8	7							0

ERR	Error Flag	Set by the controller to indicate an error condition. The command data field contains a 16 bit error code when ERR is set.
RPL	Reply Flag	Set by the controller when replying to a command.
CPL	Complete Flag	Set by the controller to indicate that a command was completed.
UF	Unused Flag	For compatibility this flag should be passed to the controller always as 0. It is currently never touched by the controller.
ID3:0	Port Identifier	A zero base ID used to specify 1 of 16 devices (e.g. UARTs).
C7:0	Command Code	The command code defining the meaning of the message (e.g. WRITE_DATA).
D15:00	Command Data	The contents of this field is dependent of the command code of the message. For example it could contain the size of the data belonging to the message or it could be a parameter itself.

The command word is defined as a union of type EKF_IMR_COMMAND in the C header “ekf960if.h”.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Exchanging Messages With The Controller

When requesting a service from the controller, this is done by the following sequence:

1. Set up a command word with all flags cleared in a local variable.
2. Copy parameter or data to the corresponding area of the MU (if any).
3. Write the command word to the controller's IMR0.

The last step will request an interrupt on the controller that is servicing the message.

The controller will always answer to a message that was send before, even if the message contains an unknown command code or bogus parameters. On reply do the following:

1. Check bit 0 of OISR for an MU interrupt request.
2. Read the contents of OMR0 to a local variable. This contains the reply to the message.
3. Clear the IRQ by writing RP_MU_OISR_MSG0 to OISR.
4. Check the command word replied, especially the flags are giving information about the status of the requested command.

Do step 3 as quickly as possible since this allows the controller to reuse the MU for other messages.

The controller does not change the command code and port ID in the reply message, thus it is easy for the driver to identify the action to which the reply belongs.

The controller sets certain flags in the reply message depending on the state of the requested command:

- The reply flag is set to reflect that this message is the answer to a command requested before, i.e. this is a reply message.
- The complete flag is set when all necessary actions are done by the controller, i.e. the corresponding command is complete. This message is a complete message.
- The error flag is set when something was wrong. This includes a bad command code, bad or missing parameters and so on. The command data field of the command word contains a 16-bit error code that specifies the reason of the failure. Error codes are defined in the C header file "ekf960if.h".

The controller may send a message in response to a message sent by the host. In this case the RPL flag is set in the reply message. If the requested command could be completed immediately, e.g. setting the baud rate of a port, the CPL flag is also set.

On the other side, when for example the host requests to transmit data over a serial port, the reply message will have the CPL flag cleared, because the data transmission will last a while. In that case the controller sends a separate complete message with CPL flag set, but RPL flag cleared, as soon as the controller is able to receive new transmit data. Note that this complete message will arrive asynchronously at any time.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

When sending a message to the controller it is important to note, that neither the contents of IMR0 nor the parameter or data buffers should be altered until the controller responds by writing a reply message to OMR0. This is best done by using a semaphore flag that indicates, when set, that the MU is currently in use. Before writing to the MU test and set the flag. When the controller responds with a reply message (RPL flag set), the semaphore can be cleared.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Data Structures Used By The Interface

The data structures used by the interface are explained in the following section. They and their possibly corresponding definitions can be found in the C header “ekf960if.h”.

EKF_MU_IO_BUFFER

A structure that describes the I/O buffers used for read and write data:

```
typedef struct _EKF_MU_IO_BUFFER
{
        UINT32      CharsInBuffer;
        UCHAR       Buffer[EKF_MU_IO_BUFFER_SIZE];
} EKF_MU_IO_BUFFER, *PEKF_MU_IO_BUFFER;
```

CharsInBuffer:

This is the number of bytes currently in the buffer. This field is maintained by the controller and should never be written by the host.

Buffer:

This is the buffer area itself. The buffer is organized in different manner dependent on its function.

As a write buffer, it is used as a simple linear buffer that is filled by the host and read by the controller from the start of the buffer with incrementing addresses. For the next write, data is filled to the start of the buffer again.

A read buffer is organized in a ring. The controller maintains a fill pointer that is incremented until the buffer is full. If the top of the buffer is reached, the fill pointer wraps to the buffer start. The fill pointer is invisible to the host. An empty pointer, administrated by the host, is used to read data from the buffer. The empty pointer read as many bytes as available in the buffer and remains at that position after the read is complete. The next read starts at that position. If the top of the buffer is reached, the empty pointer wraps to the buffer start. The amount of data can be detected by checking the CharsInBuffer field.

EKF16550_CHARS

A structure that contains special characters used for serial ports:

```
typedef struct _EKF16550_CHARS
{
        UCHAR EofChar;
        UCHAR ErrorChar;
        UCHAR BreakChar;
        UCHAR EventChar;
```

```
    UCHAR XonChar;
    UCHAR XoffChar;
} EKF16550_CHARS, *PEKF16550_CHARS;
```

EofChar:
End Of File character (currently not used).

ErrorChar:
This character, when enabled, is placed in the stream of received characters on error conditions like buffer overflow, frame errors and so on.

BreakChar:
This character, when enabled, is placed in the stream of received characters when a break condition was detected.

EventChar:
When enabled, a message is sent by the controller to the host, if this character was received by the port.

XonChar:
Defines the XON character that resumes an earlier stopped data transmission if XON/XOFF flow control is enabled.

XoffChar:
Defines the XOFF character that stops data transmission if XON/XOFF flow control is enabled.

EKF16550_HANDFLOW

A structure that contains all the stuff needed to setup hard- and software handshake for serial ports:

```
typedef struct _EKF16550_HANDFLOW
{
    UINT32 ControlHandShake;
    UINT32 FlowReplace;
    INT32 XonLimit;
    INT32 XoffLimit;
} EKF16550_HANDFLOW, *PEKF16550_HANDFLOW;
```

ControlHandShake:
A set of flags that defines the modem lines that are used for flow control:
EKF16550_DTR_HANDSHAKE:
Use the modem signal DTR for input flow control. The DTR line is cleared by the controller if the receive buffer reaches the programmed high water mark. See also description of XonLimit and XoffLimit.
EKF16550_CTS_HANDSHAKE:
EKF16550_DCD_HANDSHAKE:

EKF16550_DSR_HANDSHAKE:
Use the modem signal CTS, DCD or DSR respectively for output flow control. If the corresponding modem line(s) found as cleared, the controller will hold data transmission.
EKF16550_DSR_SENSITIVITY:
Ignore any character arriving when the DSR line is not set.
EKF16550_ERROR_ABORT:
If there exists an error condition the controller will send an error message to the host to indicate this.

FlowReplace:
A set of flags defining flow control stuff:
EKF16550_AUTO_TRANSMIT:
Use the XON/XOFF protocol based flow control for output. The reception of the XoffChar will stop data transmission until the XonChar is received (see also structure EKF16550_CHARS).
EKF16550_AUTO_RECEIVE:
Use the XON/XOFF protocol based flow control for input. The XoffChar is send by the controller if the receive buffer reaches the programmed high water mark. If the receive buffer falls below the programmed low water mark, the XonChar is send. See also description of XonLimit and XoffLimit and of structure EKF16550_CHARS.
EKF16550_ERROR_CHAR:
If set, the ErrorChar is placed in the stream of received characters on error conditions like buffer overflow, frame errors and so on. See also description of structure EKF16550_CHARS.
EKF16550_NULL_STRIPPING:
If set, the reception of a NULL character is ignored.
EKF16550_BREAK_CHAR:
If set, the BreakChar is placed in the stream of received characters when a break condition was detected. See also description of structure EKF16550_CHARS.
EKF16550_RTS_HANDSHAKE:
Use the modem signal RTS for input flow control. The RTS line is cleared by the controller if the receive buffer reaches the programmed high water mark. See also description of XonLimit and XoffLimit.

XonLimit:
When there are less than XonLimit number of characters in the read buffer the controller will perform all flow control that the host has enabled so that the sender will resume sending characters.

XoffLimit:
When there are more characters than (BufferSize - XoffLimit) in the read buffer then the controller will perform all flow control that the host has enabled so that the sender will stop sending characters.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

SJA1000_ACCEPTANCE

A structure that is used to set the frame acceptance code, mask and mode of a CANbus device to build an acceptance filter. See the data sheet of the SJA1000 stand-alone CANbus controller for details on acceptance filter programming. The library “ekf960si1.lib” that is delivered with source files also contains a function to build an acceptance filter.

```
typedef struct _SJA1000_ACCEPTANCE
{
        UINT8 code0;
        UINT8 code1;
        UINT8 code2;
        UINT8 code3;
        UINT8 mask0;
        UINT8 mask1;
        UINT8 mask2;
        UINT8 mask3;
        UINT8 singleFilter;
        UINT8 pad[3];
} SJA1000_ACCEPTANCE, *PSJA1000_ACCEPTANCE;
```

code0-3:

These 4 bytes define the acceptance code of the acceptance filter. The value of code0 is written to the register ACR0 of the SJA1000 CANbus controller, code1 to ACR1 and so on.

mask0-3:

These 4 bytes define the acceptance mask of the acceptance filter. The value of mask0 is written to the register AMR0 of the SJA1000 CANbus controller, mask1 to AMR1 and so on.

singleFilter:

This boolean, when set, flags that a single acceptance filter configuration should be used. If cleared, a dual filter configuration is used.

pad:

For alignment purposes.

EKF_INIT_PARAMS_CAN

A structure that describes the initialization parameter block passed with the initialization message CMDIMR_INIT for a CANbus device.

```
typedef struct
{
        UINT32                  ClockRate;
        UINT32                  BaudRate;
        SJA1000_ACCEPTANCE      Acceptance;
```

```
    UCHAR                   pad[12];
} EKF_INIT_PARAMS_CAN, *PEKF_INIT_PARAMS_CAN;
```

ClockRate:
Base clock of the CANbus controller in Hz.

BaudRate:
Initial baud rate in bits per second.

Acceptance:
Structure containing the initial acceptance filter (see description of structure SJA1000_ACCEPTANCE).

pad:
For alignment purposes.

EKF_INIT_PARAMS_SERIAL

A structure that describes the initialization parameter block passed with the initialization message CMDIMR_INIT for a serial device.

```
typedef struct _EKF_INIT_PARAMS_SERIAL
{
    UINT32              RxFifoTrigger;
    UINT32              TxFifoAmount;
    UINT32              ClockRate;
    UINT32              BaudRate;
    UINT32              BufferSizePt8;
    UINT8               LineControl;
    UINT8               ValidDataMask;
    UINT16              pad1;
    EKF16550_CHARS      SpecialChars;
    EKF16550_HANDFLOW   HandFlow;
} EKF_INIT_PARAMS_SERIAL, *PEKF_INIT_PARAMS_SERIAL;
```

RxFifoTrigger:
Receiver FIFO interrupt trigger level definition:
FCR_RXTRIG_1: request interrupt on each character in the FIFO,
FCR_RXTRIG_4: request interrupt on 4 characters in the FIFO,
FCR_RXTRIG_8: request interrupt on 8 characters in the FIFO,
FCR_RXTRIG_14: request interrupt on 14 characters in the FIFO.

TxFifoAmount:
Size of the transmit FIFO in bytes (max. 16).

ClockRate:
Base clock of the serial controller in Hz.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

BaudRate:
Initial baud rate in bits per second.

BufferSizePt8:
This defines a high water mark that can be used to send an event message to the host if reached. It is by default set to 80% of the receive buffer size.

LineControl:
Initial value of the line control register of the serial controller:
LCR_CHAR_LEN_5: character size is 5 bit,
LCR_CHAR_LEN_6: character size is 6 bit,
LCR_CHAR_LEN_7: character size is 7 bit,
LCR_CHAR_LEN_8: character size is 8 bit,
LCR_STOP_BIT_1: number of stop bits is 1,
LCR_STOP_BIT_2: number of stop bits is 1.5 (character size 5) or 2 (character size 6-8),
LCR_PAR_ENA: enable parity check and generation,
LCR_PAR_ODD: use odd parity if enabled,
LCR_PAR_EVEN: use even parity if enabled,
LCR_PAR_FORCED: if enabled force parity bit to 1 if PAR_ODD is also chosen, else force parity bit to 0,
LCR_BREAK_ENA: enable break condition (TX line is forced low).

ValidDataMask:
Received data is logically and'ed with this mask. ValidDataMask defaults to 0xFF.

pad1:
For alignment purposes.

SpecialChars:
Structure containing the initial special characters (see description of structure EKF16550_CHARS).

HandFlow:
Structure containing the initial handshake definitions (see description of structure EKF16550_HANDFLOW).

EKF16550_STATUS

A structure that is used to get the current error and general status of a serial port.

```
typedef struct _EKF16550_STATUS
{
    UINT32 Errors;
    UINT32 HoldReasons;
    UINT32 AmountInInQueue;
    UINT32 AmountInOutQueue;
} EKF16550_STATUS, *PEKF16550_STATUS;
```


EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Errors:
A set of flags that reflect the possible errors occurred on a serial port:
EKF16550_ERROR_BREAK: a break condition was detected,
EKF16550_ERROR_FRAMING: a framing error was detected,
EKF16550_ERROR_OVERRUN: an overrun of the serial controller's internal receiver FIFO occurred,
EKF16550_ERROR_BUFFEROVERRUN: an overrun of the read ring buffer maintained by the firmware occurred,
EKF16550_ERROR_PARITY: a parity error was detected.

HoldReasons:
A set of flags that reflects the reasons why a port could be holding:
EKF16550_TX_WAITING_FOR_CTS
EKF16550_TX_WAITING_FOR_DSR
EKF16550_TX_WAITING_FOR_DCD
EKF16550_TX_WAITING_FOR_XON
EKF16550_TX_WAITING_XOFF_SENT
EKF16550_TX_WAITING_ON_BREAK
EKF16550_RX_WAITING_FOR_DSR

AmountInInQueue:
The number of bytes that reside currently in the port's read ring buffer.

AmountInOutQueue:
The number of bytes that reside currently in the port's write buffer.

SJA1000_STATUS

A structure that is used to get the current error and general status of a CANbus port.

```
typedef struct _SJA1000_STATUS
{
    UINT32 Errors;
    UINT8 LastErrorCapture;
    UINT8 LastArbitLostCapture;
    UINT8 pad[2];
    UINT32 AmountInInQueue;
    UINT32 AmountInOutQueue;
} SJA1000_STATUS, *PSJA1000_STATUS;
```

Errors:
A set of flags that reflect the possible errors occurred on a CANbus port:
SJA1000_ERROR_BUSERROR: a bus error on the CANbus was detected,
SJA1000_ERROR_ARBITLOST: an arbitration was lost by the CANbus controller,
SJA1000_ERROR_WRITE_FRAME: a frame with bad format was passed on a WRITE_DATA command,

SJA1000_ERROR_FIFOOVERRUN: an overrun of the CANbus controller's internal receiver FIFO occurred,
SJA1000_ERROR_BUFFEROVERRUN: an overrun of the read ring buffer maintained by the firmware occurred,
SJA1000_ERROR_BUSOFF: a bus-off event on the CANbus occurred.

LastErrorCapture:
This reflects the contents of the CANbus controller's Error Capture Register (ECC) at the moment when a bus error was detected. This can give additional information about the type and location of the error. See the data sheet of the SJA1000 CANbus controller for a description of the ECC.

LastArbitLostCapture:
This reflects the contents of the CANbus controller's Arbitration Lost Capture Register (ALC) at the moment when an arbitration lost was detected. This can give additional information about the bit position where the lost occurred. See the data sheet of the SJA1000 CANbus controller for a description of the ALC.

pad:
For alignment purposes.

AmountInInQueue:
The number of bytes that reside currently in the port's read ring buffer.

AmountInOutQueue:
The number of bytes that reside currently in the port's write buffer.

EKF16550_PERF_STATS

A structure that is used to get the current performance statistic counter values of a serial port.

```
typedef struct _EKF16550_PERF_STATS
{
    UINT32 ReceivedCount;
    UINT32 TransmittedCount;
    UINT32 FrameErrorCount;
    UINT32 SerialOverrunErrorCount;
    UINT32 BufferOverrunErrorCount;
    UINT32 ParityErrorCount;
} EKF16550_PERF_STATS, *PEKF16550_PERF_STATS;
```

ReceivedCount:
The number of characters received successfully.

TransmittedCount:
The number of characters transmitted successfully.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

FrameErrorCount:
The number of framing errors detected by the serial controller.

SerialOverrunErrorCount:
The number of overruns of the serial controller's internal receive FIFO.

BufferOverrunErrorCount:
The number of overruns of the read ring buffer maintained by the firmware.

ParityErrorCount:
The number of parity errors detected by the serial controller.

SJA1000_PERF_STATS

A structure that is used to get the current performance statistic counter values of a CANbus port.

```
typedef struct _SJA1000_PERF_STATS
{
    UINT32 ReceivedCount;
    UINT32 TransmittedCount;
    UINT32 BusErrorCount;
    UINT32 ArbitLostCount;
    UINT32 FifoOverrunErrorCount;
    UINT32 BufferOverrunErrorCount;
    UINT32 ErrorWarningIrqCount;
    UINT32 ErrorPassiveIrqCount;
    UINT32 WakeUpIrqCount;
    UINT32 TransmitErrorCount;
} SJA1000_PERF_STATS, *PSJA1000_PERF_STATS;
```

ReceivedCount:
The number of frames received successfully.

TransmittedCount:
The number of frames transmitted successfully.

BusErrorCount:
The number of bus errors detected by the CANbus controller.

ArbitLostCount:
The number of arbitrations that were lost by the CANbus controller.

FifoOverrunErrorCount:
The number of overruns of the CANbus controller's internal receive FIFO.

BufferOverrunErrorCount:
The number of overruns of the read ring buffer maintained by the firmware.

ErrorWarningIrqCount:
The number of Error Warning Interrupts. This count is incremented on every bus status change (bus-on to bus-off or vice versa).

ErrorPassiveIrqCount:
The number of Error Passive Interrupts. This count is incremented when the CAN controller reaches the error passive status (at least one of the SJA1000 internal error counters reached the level of 127) or leaves the error passive status.

WakeUpIrqCount:
This count is incremented on every CAN controller wake-up.

TransmitErrorCount:
The number of frames whose transmissions failed.

EKF_DOWNLOAD_PARAMS

A structure that is used to pass download parameters to the firmware when downloading a new firmware binary.

```
typedef struct _EKF_DOWNLOAD_PARAMS
{
        UINT32                  byteCount;
        UINT32                  flashOffset;
} EKF_DOWNLOAD_PARAMS, *PEKF_DOWNLOAD_PARAMS;
```

byteCount:
The number of bytes that are following this structure (the real download data). This number should be aligned to INT32, i.e. the last two bits should be 0.

flashOffset:
The offset within the flash ROMS in bytes where the download data block should written to. This offset should be aligned to INT32, i.e. the last two bits should be 0.

Command Set

The following section will give a detailed description of the commands implemented on the EKF Intelligent I/O Controllers. Note that some of these commands are understood only by specific types of controllers.

The command codes and their possibly corresponding data structures and definitions can be found in the C header “ekf960if.h”.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Version Of The Firmware: CMDIMR_VERSION_GET

Description: This command is used to get the version of the firmware currently running on the controller.

Command Code: 1.

Port Identifier: Don't Care.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0001 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	V	V	V	V	V	V	V	V	E	E	E	E	E	E	E	E
31			28	27			24	23							16	15							8	7							0

The controller returns an 8-bit version number in bits 15:08 and an 8-bit edition number in bits 07:00 with the RPL and CPL flags set. The version and edition information can be gotten by the .version.version and the .version.edition members of the command word union EKF_IMR_COMMAND.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Initialize A Port: CMDIMR_INIT

Description: This command is used to initialize a port specified by the port identifier. The port is brought in a quiescent state. Some device parameters are set to the values passed by the initialization parameter block with this message.

Command Code: 2.

Command Data: Size of the initialization parameter block in bytes.

Parameters: A set of parameters is passed in the inbound parameter area. These parameters are dependent on the port type that is initialized. Serial port initialization parameters are described by the structure EKF_INIT_PARAMS_SERIAL, CANbus ports use the structure EKF_INIT_PARAMS_CAN.

Data: None.

Command Word: 0x0Y02 AAAA

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	0	0	0	1	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	0	0	0	1	0	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the size of the port's data buffer in bytes in bits 15:00 with the RPL and CPL flags set. The buffer size can be gotten by the .init.bufferSize member of the command word union EKF_IMR_COMMAND.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: An INIT command must be executed after the board got a hardware reset.

See section "**Port Arrangement**" for the mapping of different port types and port identifier on the I/O boards. For a description of the initialization

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

parameter structures for the different types of ports see section "**Data Structures Used By The Interface**".

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Deinitialize A Port: CMDIMR_DEINIT

Description: This command is used to deinitialize a port specified by the port identifier. The port is brought in a quiescent state, i.e. all of its interrupts will be disabled.

Command Code: 3.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y03 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Open A Port: CMDIMR_OPEN

Description: This command is used to open a port specified by the port identifier. The port is initialized, the receive buffer is purged, i.e. the fill pointer is set to begin of the read buffer, the performance statistics are cleared and finally the port IRQs will be enabled.

Command Code: 4.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y04 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: An OPEN command must be executed before any port I/O or I/O control is possible.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Close A Port: CMDIMR_CLOSE

Description: This command is used to close a port specified by the port identifier. First the port interrupts will be disabled. On serial ports the controller then waits for the transmission of the data that currently reside in the transmit FIFO of the UART. If programmed for XON/XOFF flow control, an XON character is sent when reception was held before by sending XOFF. After that the controller waits 10 character times before clearing the DTR and RTS lines.

Command Code: 5.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y05 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	X	0	Y	Y	Y	Y	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

On serial ports the controller returns with the RPL flag set. A separate complete message is sent by the controller with CPL flag set as soon as all operations have been completed.

On CANbus ports the controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: See section "**Port Arrangement**" for the mapping of different port types and port identifier on the I/O boards.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Write Data To Port: CMDIMR_WRITE_DATA

Description: This command is used to write a block of data to a port specified by the port identifier. The write data is copied to the begin of the write buffer of the corresponding port.

Command Code: 6.

Command Data: Size of the data block in bytes written to the write buffer.

Parameters: None.

Data: The data block to write is passed in the write buffer of the port.

Command Word: 0x0Y06 AAAA

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	0	0	1	1	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	0	0	Y	Y	Y	Y	0	0	0	0	0	1	1	0	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the number of bytes of the write data that was really accepted by the controller in bits 15:00 and the RPL flag set. A separate complete message is sent by the controller with CPL flag set as soon as the write operation has been completed.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: To send data on a CANbus port, a complete frame including frame information field, TX identifier and TX data must be supplied to the write buffer. The controller scans the frame information for the frame format bit and the data length code to get information about the frame to send. It is possible to copy as many frames to the write buffer as possible and send them with one WRITE command. The controller will return an error if a bad frame was passed and ignore any data in the write buffer behind the bad frame.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

A new WRITE command should issued only if a previous write was completed. Otherwise the contents of the old write data could be corrupted.

Do not write the number of the bytes in the write buffer directly to the CharsInBuffer field of the write buffer. This field is managed by the controller only, although reading this field is possible any time.

If a byte count greater than the write buffer size was passed with the command the controller will truncate the number to the write buffer size and return this amount. However, data corruption of a neighbored port will happen if the write buffer limit is exceeded.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Kill Current Write: CMDIMR_KILL_WRITE

Description: This command is used to kill a write currently in progress on a port specified by the port identifier.

Command Code: 7.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y07 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	0	0	1	1	1	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the number of bytes that still remained in the write buffer when the write was killed in bits 15:00 with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Read Data From Port: CMDIMR_RED_DATA

Description: This command is used to tell the controller how many data the host has red from a port's read buffer. The port identifier specifies the port. The reported amount of data was copied from the read buffer of the corresponding port before this command.

Command Code: 8.

Command Data: Size of the data block red from the read buffer before.

Parameters: None.

Data: None.

Command Word: 0x0Y08 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	0	1	0	0	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	0	1	0	0	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set. The contents of bits 15:00 in the reply word is unchanged.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: The read buffer is organized as a ring buffer. Its handling is described in the section "**EKF_MU_IO_BUFFER**".

A RED_DATA command terminates a READ_NEED request set before.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Set Amount Needed For Read: CMDIMR_READ_NEED

Description: This command is used to set the amount of data the host needs for a read. The port is specified by the port identifier. As soon as the requested number of bytes are available in the read buffer, the controller sends a message to the host.

Command Code: 9.

Command Data: Number of bytes needed for the read.

Parameters: None.

Data: None.

Command Word: 0x0Y09 AAAA

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	0	1	0	0	1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	X	0	Y	Y	Y	Y	0	0	0	0	1	0	0	1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL flag set. If there is already enough data in the read buffer, the CPL flag is set also. Otherwise a separate complete message is sent as soon as the requested amount of data was received by the port. The contents of bits 15:00 in the reply word is unchanged.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Purge The Read Buffer: CMDIMR_PURGE_READ

Description: This command is used to purge the read buffer of a port specified by the port identifier. The number of bytes in the read buffer is cleared and the fill pointer reset to the buffer start.

Command Code: 10.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y0A 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: This command is useful to synchronize the read ring buffer pointers maintained by the firmware and the host. Therefore it is necessary for the host to reset its empty pointer to the read buffer start when executing this command.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Setup Port Baud Rate: CMDIMR_SET_BAUD

Description: This command is used to setup the transmission speed of a port specified by the port identifier.

Command Code: 16.

Command Data: Size of the baud rate parameter in bytes.

Parameters: The baud rate in bits per second as an unsigned long is passed in the inbound parameter area.

Data: None.

Command Word: 0x0Y10 0004

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: The controller accepts all well known baud rates. Furthermore any baud rate that can be programmed with a resulting error of less than 1% can be chosen.

The baud rate chosen by this command is retained across openings and closings.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Baud Rate: CMDIMR_GET_BAUD

Description: This command is used to get the baud rate currently set on a port specified by the port identifier.

Command Code: 17.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y11 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns the size in bytes of the baud rate parameter (i.e. sizeof(long)) in bits 15:00 with the RPL and CPL flags set. The baud rate currently set is returned as an unsigned long in the outbound parameter area.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Setup Line Control: CMDIMR_SET_LINE_CTL

Description: This command is used to setup the line control value of a port specified by the port identifier.

Command Code: 18.

Command Data: The line control value in bits 07:00.

Parameters: None.

Data: None.

Command Word: 0x0Y12 00AA

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: The line control word passed with this command is written to the Line Control Register (LCR) of the serial controller 16550. See also section "**EKF_INIT_PARAMS_SERIAL**" for a description of the line control word.

The line control value chosen by this command is retained across openings and closings.

This command is acceptable for serial ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Line Control: CMDIMR_GET_LINE_CTL

Description: This command is used to get the line control value currently set on a port specified by the port identifier.

Command Code: 19.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y13 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	B	B	B	B	B	B	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the line control value currently set in bits 07:00 with the RPL and CPL flags set. The line control value can be gotten by the .param8.data member of the command word union EKF_IMR_COMMAND.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: This command really returns the current value of the Line Control Register (LCR) of the serial controller 16550. See also section "**EKF_INIT_PARAMS_SERIAL**" for a description of the line control word.

This command is acceptable for serial ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Set Modem Line DTR: CMDIMR_SET_DTR

Description: This command is used to set the modem line *Data Terminal Ready* (DTR) of a port specified by the port identifier.

Command Code: 20.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y14 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: This command is acceptable for serial ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Clear Modem Line DTR: CMDIMR_CLR_DTR

Description: This command is used to clear the modem line *Data Terminal Ready* (DTR) of a port specified by the port identifier.

Command Code: 21.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y15 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: This command is acceptable for serial ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Set Modem Line RTS: CMDIMR_SET_RTS

Description: This command is used to set the modem line *Request To Send* (RTS) of a port specified by the port identifier.

Command Code: 22.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y16 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: This command is acceptable for serial ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Clear Modem Line RTS: CMDIMR_CLR_RTS

Description: This command is used to clear the modem line *Request To Send* (RTS) of a port specified by the port identifier.

Command Code: 23.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y17 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: This command is acceptable for serial ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Setup Flow Control: CMDIMR_SET_HANDFLOW

Description: This command is used to setup the flow control values of a port specified by the port identifier.

Command Code: 24.

Command Data: Size of the flow control parameter structure in bytes.

Parameters: The flow control parameter structure EKF16550_HANDFLOW is passed in the inbound parameter area.

Data: None.

Command Word: 0x0Y18 AAAA

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	1	0	0	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	1	0	0	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: See section "**EKF16550_HANDFLOW**" for a description of the flow control parameter structure.

The flow control settings chosen by this command are retained across openings and closings.

This command is acceptable for serial ports only.

If the EKF16550_ERROR_ABORT option is enabled in the member ControlHandShake of the flow control parameter structure an error message is sent to the host if at least one error condition exists for the port. The error message has the following appearance:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	1	0	Y	Y	Y	Y	0	0	0	0	1	1	0	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

The error message has the command code 12 and passes the error(s) happened in bits 15:00 with the CPL flag set. The port ID specifies the port where the error condition occurred. A description of the error flags can be obtained from section “**EKF16550_STATUS**”.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Flow Control: CMDIMR_GET_HANDFLOW

Description: This command is used to get the flow control parameter currently set on a port specified by the port identifier.

Command Code: 25.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y19 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	1	0	0	1	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the size in bytes of the flow control parameter structure in bits 15:00 with the RPL and CPL flags set. The flow control parameters currently set are returned as a structure of type EKF16550_HANDFLOW in the outbound parameter area.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: See section "**EKF16550_HANDFLOW**" for a description of the flow control parameter structure.

This command is acceptable for serial ports only.

This command is currently not supported.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Modem Status: CMDIMR_GET_MODEMSTAT

Description: This command is used to get the current modem line setting of a port specified by the port identifier.

Command Code: 26.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y1A 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	B	B	B	B	B	B	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the current modem line status in bits 07:00 with the RPL and CPL flags set. The modem line status can be gotten by the .param8.data member of the command word union EKF_IMR_COMMAND.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: This command really returns the current value of the modem status register (MSR) of the serial controller 16550:

- Bit 7: Status of modem line *Data Carrier Detect* (DCD)
- Bit 6: Status of modem line *Ring Detect Indicator* (RI)
- Bit 5: Status of modem line *Data Set Ready* (DSR)
- Bit 4: Status of modem line *Clear To Send* (CTS)

This command is acceptable for serial ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get DTR/RTS Status: CMDIMR_GET_DTRRTS

Description: This command is used to get the current setting of the modem lines *Data Terminal Ready* (DTR) and *Request To Send* (RTS) of a port specified by the port identifier.

Command Code: 27.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y1B 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the current modem line status of DTR in bit 00 and RTS in bit 01 with the RPL and CPL flags set. The DTR/RTS line status can be gotten by the .param8.data member of the command word union EKF_IMR_COMMAND.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: This command is acceptable for serial ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Setup Special Characters: CMDIMR_SET_CHARS

Description: This command is used to setup the special characters of a port specified by the port identifier.

Command Code: 28.

Command Data: Size of the special characters structure in bytes.

Parameters: The special characters structure EKF16550_CHARS is passed in the inbound parameter area.

Data: None.

Command Word: 0x0Y1C AAAA

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	1	1	0	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	1	1	0	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: See section "**EKF16550_CHARS**" for a description of the special characters structure.

The special characters chosen by this command are retained across openings and closings.

This command is acceptable for serial ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Special Characters: CMDIMR_GET_CHARS

Description: This command is used to get the special characters currently set on a port specified by the port identifier.

Command Code: 29.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y1D 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	1	1	0	1	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the size in bytes of the special characters structure in bits 15:00 with the RPL and CPL flags set. The special characters currently set are returned as a structure of type EKF16550_CHARS in the outbound parameter area.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: See section "**EKF16550_CHARS**" for a description of the special characters structure.

This command is acceptable for serial ports only.

This command is currently not supported.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Port Status: CMDIMR_GET_COMMSTAT

Description: This command is used to get the current status of a port specified by the port identifier. This includes the number of characters in the read and write buffer, the error status and so on.

Command Code: 30.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y1E 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	1	1	1	0	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the size in bytes of the port status structure in bits 15:00 with the RPL and CPL flags set. The port status is returned as a structure of type EKF16550_STATUS for serial ports and SJA1000_STATUS for CANbus ports in the outbound parameter area.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: See section "**EKF16550_STATUS**" for a description of the port status structure for serial ports and section "**SJA1000_STATUS**" for CANbus ports. See also section "**Port Arrangement**" for the mapping of different port types and port identifier on the I/O boards.

The port error status word kept on the controller is cleared after this command was executed.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Setup Event Mask: CMDIMR_SET_EV_MASK

Description: This command is used to setup an event mask of a port specified by the port identifier.

Command Code: 31.

Command Data: The event mask in bits 15:00.

Parameters: None.

Data: None.

Command Word: 0x0Y1F AAAA

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	0	1	1	1	1	1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	0	1	1	1	1	1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: The event mask passed with this command contains a set a flags that may lead to an event message notifying the host that something was happened. The following events are valid for serial ports:

EKF16550_EV_RXCHAR: any character received,
EKF16550_EV_RXFLAG: received event character specified in the special characters,
EKF16550_EV_TXEMPTY: transmit queue empty,
EKF16550_EV_CTS: CTS changed state,
EKF16550_EV_DSR: DSR changed state,
EKF16550_EV_RLSD: DCD changed state,
EKF16550_EV_BREAK: break received,
EKF16550_EV_ERR: line status error occurred,
EKF16550_EV_RING: ring signal detected,
EKF16550_EV_RX80FULL: receive buffer is 80% full.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

These are the events supported on CANbus ports:

SJA1000_EV_RXFRAME: a frame received,
SJA1000_EV_BUSOFF: a bus-off event occurred,
SJA1000_EV_BUSON: a bus-on event occurred,
SJA1000_EV_ERR: an error occurred.

An event message is sent to the host if at least one of the enabled events occurred. The event message has the following appearance:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	1	0	Y	Y	Y	Y	0	0	0	0	1	0	1	1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

The event message has the command code 11 and passes the event(s) happened in bits 15:00 with CPL flag set. The port ID specifies the port where the event(s) occurred. The events correspond to the event flags described above.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Event Mask: CMDIMR_GET_EV_MASK

Description: This command is used to get the current event mask setting of a port specified by the port identifier.

Command Code: 32.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y20 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	1	0	0	0	0	0	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the current event mask in bits 15:00 with the RPL and CPL flags set. The event mask can be gotten by the .event.event member of the command word union EKF_IMR_COMMAND.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: See CMDIMR_SET_EV_MASK for a description of the event mask returned by this command.

This command is currently not supported.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Performance Statistics: CMDIMR_GET_STATS

Description: This command is used to get the current values of the performance statistic counters of a port specified by the port identifier.

Command Code: 33.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y21 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	1	0	0	0	0	1	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the size in bytes of the performance statistics structure in bits 15:00 with the RPL and CPL flags set. The statistics structure is returned as a structure of type EKF16550_PERF_STATS in the outbound parameter area.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: See section "**EKF16550_PERF_STATS**" for a description of the performance statistics structure.

This command is acceptable for serial ports only. See command CMDIMR_GET_STATS_CAN for CANbus ports.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Clear Statistics Counters: CMDIMR_CLR_STATS

Description: This command is used to clear the performance counters of a port specified by the port identifier.

Command Code: 34.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y22 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Turn Break On: CMDIMR_BREAK_ON

Description: This command is used to turn on break of a port specified by the port identifier.

Command Code: 35.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y23 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: This command is acceptable for serial ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Turn Break Off: CMDIMR_BREAK_OFF

Description: This command is used to turn off break of a port specified by the port identifier.

Command Code: 36.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y24 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: This command is acceptable for serial ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Setup Insert Mode: CMDIMR_SET_INSERT_MODE

Description: This command is used to setup the insertion of information about the line status and the modem status in the receive data stream of a port specified by the port identifier. An escape character is passed with this message. If the escape character is 0x00 then the insertion mode is turned off.

Command Code: 38.

Command Data: The escape character in bits 07:00.

Parameters: None.

Data: None.

Command Word: 0x0Y26 AAAA

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: The information inserted always starts with the escape character passed with this message and a following character describing the type of event happened.

On receive error the escape character and the character EKF16550_LSRMST_LSR_DATA are placed in the read ring buffer followed by the contents of the *Line Status Register* (LSR) of the serial controller and the character received.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

If no receive data was available when the receive error occurred, the escape character, EKF16550_LSRMST_LSR_NODATA and the contents of LSR is placed in the read ring buffer.

On changes of the modem lines the escape character, the character EKF16550_LSRMST_MST and the contents of the *Modem Status Register* (MSR) is placed in the read ring buffer.

The reception of the escape character itself is indicated by the insertion of the sequence escape character, EKF16550_LSRMST_ESCAPE, escape character.

This command is acceptable for serial ports only.

Erase Firmware Flash ROMs: CMDIMR_FLASH_ERASE

Description: This command is used to erase the firmware flash ROMs on the controller.

Command Code: 40.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y28 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	0	0	Y	Y	Y	Y	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL flag set. A separate complete message is sent by the controller with CPL flag set as soon as the erasure operation has been completed. If the erasure failed the complete message comes with ERR flag set and an error code is passed in bits 15:00 of the complete message.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: Absolutely caution should be given when executing this command. Use makes sense only if a new firmware binary is available that is downloaded to the firmware flash ROMs with the command CMDIMR_FLASH_WRITE after erasing. **The board is no more functional, if a hardware reset occurred while or after erasing.**

The host may use any port ID that is valid for the corresponding board when calling the flash ROM erasure command. It is not necessary to open the port used before.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Write Firmware Flash ROMs: CMDIMR_FLASH_WRITE

Description: This command is used to write a block of new data to the firmware flash ROMs on the controller.

Command Code: 41.

Command Data: Size of the download parameter structure in bytes.

Parameters: The download parameters structure EKF_DOWNLOAD_PARAMS followed by the download data is passed in the download area.

Data: None.

Command Word: 0x0Y29 AAAA

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	1	0	0	1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	0	0	Y	Y	Y	Y	0	0	1	0	1	0	0	1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL flag set. A separate complete message is sent by the controller with CPL flag set as soon as the write operation has been completed. If the write failed the complete message comes with ERR flag set and an error code is passed in bits 15:00 of the complete message.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: Absolutely caution should be given when executing this command. Use makes sense only if a new firmware binary is available that is downloaded to the firmware flash ROMs. Erasure of the old firmware with the command CMDIMR_FLASH_ERASE is necessary before. **The board is no more functional, if a hardware reset occurred while writing the new firmware binary.**

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

The host may use any port ID that is valid for the corresponding board when calling the flash ROM write command. It is not necessary to open the port used before.

The size of the download data block is limited to EKF_MU_DOWNLOAD_BUFFER_SIZE minus the size of the structure EKF_DOWNLOAD_PARAMS. Therefore it is necessary to split and download the firmware in several blocks.

After all blocks of the new firmware binary are written to the flash ROMs a board hardware reset must be supplied to start the new firmware.

See section "**EKF_DOWNLOAD_PARAMS**" for a description of the download parameter structure.

Read Firmware Flash ROMs: CMDIMR_FLASH_READ

Description: This command is used to read a block of data from the firmware flash ROMs on the controller.

Command Code: 42.

Command Data: Size of the download parameter structure in bytes.

Parameters: The download parameters structure EKF_DOWNLOAD_PARAMS is passed in the download area.

Data: None.

Command Word: 0x0Y2A AAAA

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	1	0	1	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	0	0	Y	Y	Y	Y	0	0	1	0	1	0	1	0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL flag set. A separate complete message is sent by the controller with CPL flag set as soon as the read operation has been completed.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: The host may use any port ID that is valid for the corresponding board when calling the flash ROM read command. It is not necessary to open the port used before.

See section "**EKF_DOWNLOAD_PARAMS**" for a description of the download parameter structure.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Setup Acceptance Filter: CMDIMR_SET_ACCEPTANCE

Description: This command is used to setup the acceptance filter of a port specified by the port identifier.

Command Code: 43.

Command Data: Size of the acceptance filter structure in bytes.

Parameters: The acceptance filter structure SJA1000_ACCEPTANCE is passed in the inbound parameter area.

Data: None.

Command Word: 0x0Y2B AAAA

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	1	0	1	1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	1	0	1	0	1	1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: See section "**SJA1000_ACCEPTANCE**" for a description of the acceptance filter structure.

The acceptance filter settings chosen by this command are retained across openings and closings.

This command is acceptable for CANbus ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Acceptance Filter: CMDIMR_GET_ACCEPTANCE

Description: This command is used to get the current acceptance filter settings of a port specified by the port identifier.

Command Code: 44.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y2C 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	1	0	1	1	0	0	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the size in bytes of the acceptance filter structure in bits 15:00 with the RPL and CPL flags set. The acceptance filter is returned as a structure of type SJA1000_ACCEPTANCE in the outbound parameter area.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: See section "**SJA1000_ACCEPTANCE**" for a description of the acceptance filter structure.

This command is acceptable for CANbus ports only.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get Statistics Counters: CMDIMR_GET_STATS_CAN

Description: This command is used to get the current values of the performance counters of a port specified by the port identifier.

Command Code: 45.

Command Data: None.

Parameters: None.

Data: None.

Command Word: 0x0Y2D 0000

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data															
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	1	0	1	1	0	1	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
31			28	27			24	23							16	15							8	7							0

The controller returns the size in bytes of the performance statistics structure in bits 15:00 with the RPL and CPL flags set. The counter structure is returned as a structure of type SJA1000_PERF_STATS in the outbound parameter area.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: See section "**SJA1000_PERF_STATS**" for a description of the performance statistics structure.

This command is acceptable for CANbus ports only. See command CMDIMR_GET_STATS for serial ports.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Get CANbus Controller Register: CMDIMR_GET_REG_CAN

Description: This command is used to get the current value of a register from the SJA1000 CANbus controller of a CANbus port specified by the port identifier.

Command Code: 46.

Command Data: The controller's register number is passed in bits 15:08.

Parameters: None.

Data: None.

Command Word: 0x0Y2E RR00

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	1	1	1	0	R	R	R	R	R	R	R	R	0	0	0	0	0	0	0	0
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	1	0	1	1	1	0	R	R	R	R	R	R	R	R	D	D	D	D	D	D	D	D
31			28	27			24	23							16	15							8	7							0

The controller returns the current register contents in bits 07:00 with the RPL and CPL flags set. The value can be gotten by the .reg.data member of the command word union EKF_IMR_COMMAND.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: Absolutely caution should be given when using this command. In fact most of the SJA1000 CANbus controller's register could be read anytime, but for example the Interrupt Register IR may change its contents on a read access without knowledge of the firmware. Thus you should be familiarly with the function of the SJA1000 CANbus controller to avoid data lost or locking conditions.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Set CANbus Controller Register: CMDIMR_SET_REG_CAN

Description: This command is used to set a register of the SJA1000 CANbus controller of a CANbus port specified by the port identifier to a new value.

Command Code: 47.

Command Data: The controller's register number is passed in bits 15:08 and the new register value is passed in bits 07:00.

Parameters: None.

Data: None.

Command Word: 0x0Y2F RRDD

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
0	0	0	0	Y	Y	Y	Y	0	0	1	0	1	1	1	1	R	R	R	R	R	R	R	R	D	D	D	D	D	D	D	D
31			28	27			24	23							16	15							8	7							0

Reply:

Flags				Port ID				Command								Command Data													
ERR	RPL	CPL	UF	ID3	ID2	ID1	ID0	C7	C6	C5	C4	C3	C2	C1	C0	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
X	1	1	0	Y	Y	Y	Y	0	0	1	0	1	1	1	1	R	R	R	R	R	R	R	R	D	D	D	D	D	D	D	D
31			28	27			24	23							16	15							8	7							0

The controller returns with the RPL and CPL flags set.

On error the RPL, CPL and ERR flags are set and an error code is passed in bits 15:00 of the reply word.

Notes: Absolutely caution should be given when executing this command. The use makes sense only when a CANbus port is accessed exclusively with SET_REG and GET_REG commands without opening it before. Otherwise it is possible to lock the complete firmware. Thus you should be familiarly with the function of the SJA1000 CANbus controller to avoid data lost or locking conditions.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Error Codes

The error codes returned by the command messages are explained in this section. They can be found in the C header “ekf960if.h”.

EIMR_UNKNOWN_CMD
: A message was given to the controller with a command code that is unknown to the firmware.

EIMR_BAD_ID
: A message was given to the controller with a port ID number that doesn’t correspond to a port on the board. In most cases the ID is too large, e.g. there are 4 ports on the board and the ID is greater than 3 (valid are 0...3).

EIMR_BAD_COUNT
: A message was given to the controller with a byte count in the command data field that was not expected by the firmware (e.g. the size of a structure passed doesn’t match the size the firmware expected).

EIMR_BAD_BAUD
: A CMDIMR_SET_BAUD message was given to the controller that passed a baud rate not supported by the port.

EIMR_BAD_EV_MASK
: A CMDIMR_SET_EV_MASK message was given to the controller that passed a bad event mask, i.e. events that are not supported by the port.

EIMR_FLASH_ERASE
: The erasure of the firmware flash ROMs failed.

EIMR_FLASH_OFFS
: A message was given to the controller that passed a bad flash offset in the download parameter structure, i.e. one that is out of range.

EIMR_FLASH_WRITE
: Writing a block of data to the firmware flash ROMs failed.

EIMR_BAD_ESCAPE_CHAR
: A CMDIMR_SET_INSERT_MODE message was given to the controller that passed a bad escape character. The escape character must not be equal to the XON or XOFF character.

EIMR_UNSUPPORTED_CMD
: A message was given to the controller that was accepted by the firmware’s message handler but was not supported by the port.

EIMR_BAD_REG
: A message was given to the controller that passed a register number out of range.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

EIMR_BAD_FRAME
: A CMDIMR_WRITE_DATA message was given to a CANbus port on the controller that passed a frame with a bad format.

EIMR_XMIT_FAILED
: The transmission of a frame of a CANbus port failed.

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Port Arrangement

The EKF Intelligent I/O Controllers may contain one or more ports of different type. These are currently serial ports and CANbus ports. A port on a board is referenced by a port ID. A port ID has a fixed relationship to a port on a board. The following table shows the mapping of the port IDs for different boards:

Port ID Mapping

Port ID	CG1-RADIO	CU1-CHORUS	CU2-QUARTET	CX1-BAND
0	SP1	SP1	SP1	SP1
1	SP2	SP2	SP2	CAN1
2	n/a	SP3	SP3	CAN2
3	n/a	SP4	SP4	n/a
4	n/a	SP5	n/a	n/a
5	n/a	SP6	n/a	n/a
6	n/a	SP7	n/a	n/a
7	n/a	SP8	n/a	n/a
8	n/a	SP9	n/a	n/a
9	n/a	SP10	n/a	n/a
10	n/a	SP11	n/a	n/a
11	n/a	SP12	n/a	n/a
12	n/a	SP13	n/a	n/a
13	n/a	SP14	n/a	n/a
14	n/a	SP15	n/a	n/a
15	n/a	SP16	n/a	n/a

Notes:
[1]) SP: serial port
[2]) n/a: not available

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de

Additional Documentation

A detailed description of the features including the programming of the i960®RP processor, the serial controller 16C550 and the CANbus controller SJA1000 could be find in the documentation listed below.

Related Documentation

Document Title	Order Number
i960®Rx I/O Microprocessor Developer's Manual	Intel Order # 272736
i960®Rx I/O Processor Specification Update	Intel Order # 272918
i960®RP/RD I/O Processor at 3.3 Volts Data Sheet	Intel Order # 273001
i960®Jx Microprocessor User's Guide	Intel Order # 272483
16C550 UART Data Sheet	Exar, NSC, TI
SJA1000 Stand-Alone CAN Controller Data Sheet	Philips Semiconductors

Electronic information can be obtained via

http://developer.intel.com

http://www.exar.com

http://www.microsoft.com

http://www.national.com

EKF Elektronik GmbH * Philipp-Reis-Str. 4 * D-59065 HAMM (Germany)
Tel. +49 (0)2381/6890-0 * Fax. +49 (0)2381/6890-90 * E-Mail info@ekf.de * Internet http://www.ekf.de